R. MOLENWORTH

INTRODUCTION TO ASSEMBLY LANGUAGE

Introduction to Assembly Language for the TI Home Computer

by Ralph Molesworth

Edited by Steve Davis

PRESENTAZIONE

Due brevi righe per presentare questa traduzione, che é stata fatta per uso personale (e si vede...?!) e che ora tramite l'Amico e Texano Giancarlo ANTICI di Roma, autore di altre traduzioni sul TI-99/4A (1), ho deciso di mettere a disposizione degli altri Amici Texani, a cui spero possa essere di aiuto nella comprensione di questo benedetto linguaggio ASSEMBLY.

E' la mia seconda esperienza, dopo la traduzione del manuale operativo del programma di grafica "GRAPHX".

Spero che il risultato della traduzione sia leggibile e comprensibile a tutti, nonostante qualche manchevolezza che potrà apparire qua e là, ma che mi auguro non pregiudichi il suo insieme.

Ennio MEDICI
via Stazione 21
18030 BEVERA (IM)
tel. 0184/210068

(1) Libri disponibili presso Giancarlo ANTICI
via G. Cardano 170
00146 Roma

- M.S. MORLEY
  Fundamentals of TI-99/4A Assembly Language
- AMROUCHE & DIDI
  Initiation au langage Assembleur du TI-99/4A

N.B: per un salto nella numerazione la pag. 63 non esiste e si passa quindi direttamente da pag. 62 a pag. 64.

INDICE

CAPITOLO PRIMO

INTRODUZIONE

Lo scopo di questo libro, come e' implicito dal suo titolo, e' di introdurvi nell'uso del TI-99/4A con il LINGUAGGIO ASSEMBLY DEL TMS9900, e farvi iniziare a scrivere programmi in ASSEMBLY con il minimo sforzo.

I manuali forniti con i package dell'EDITOR /ASSEMBLER, (E/A) e MINI MEMORY M/M) sono soltanto dei manuali di riferimento. Essi presumono che voi abbiate una qualche conoscenza di programmazione in ASSEMBLY. Naturalmente questo crea dei problemi per i principianti. Ora questo libro puo' fungere come supplemento per darvi una migliore conoscenza generale, e la capacita' necessaria per capire e utilizzare meglio il vostro manuale dell'E/A

Al termine di questo volumetto sarete in grado di decifrare semplici programmi in LINGUAGGIO ASSEMBLY del TMS9900, e userete meglio il vostro E/A, o l'Assembler LINEA per LINEA della M/M, e i manuali di riferimento per sviluppare programmi e routines piu' complessi... almeno spero. (Nota del traduttore)

E' meglio leggere questo libro dall'inizio, perche' ogni lezione e' basata su quella precedente e gli esempi dei programmi spiegano le funzioni descritte fino a quel punto. Per sfruttare al meglio questo libro dovrete avere il software e il manuale dell'E/A. Quelli che non possiedono le periferiche necessarie per usare questo software, (Espansione di memoria 32K e disk driver) potranno usare il modulo della M/M per inserire molti esempi di programmi. In questo libro quando e' nominato l'ASSEMBLER LINEA PER LINEA, esso verra' assegnato al programma ASSEMBLER che e' fornito con la M/M. L' ASSEMBLER LINEA PER LINEA ha certe limitazioni, che vengono descritte piu' avanti, ma i concetti dei programmi ASSEMBLY sono gli stessi dell'E/A.

Alla fine di ogni sezione ci sono informazioni di riferimento al manuale dell'E/A, dove potranno essere trovate informazioni supplementari.

Prima di procedere con questo lavoro, voi dovreste prendere confidenza con l'uso del TI BASIC. Esattamente come un linguaggio BASIC varia da un computer all'altro, cosi' sono anche i linguaggi ASSEMBLY. Ma se siete in grado di imparare una forma di linguaggio, sara' semplice applicare cio' che avete imparato ad un'altro linguaggio simile. Ora voi dovrete avere un p'o' d'esperienza di programmazione con semplici programmi in TI BASIC, e EX/BASIC. Voi noterete che l'ASSEMBLY e' assai piu' potente del BASIC. Ma egli e' un linguaggio del computer, e come tutti, ha norme, discipline e termini particolari che dovranno essere imparati un p'o' per volta. Una volta imparato piu' di un paio di linguaggi del computer, le rassomiglianze tra loro saranno piu' evidenti delle differenze.

PASSARE DAL BASIC ALL'ASSEMBLY

Le istruzioni dei programmi che avete scritto fin'ora in TI BASIC, prese cosi' come sono, non hanno alcun significato per il computer. Ogni istruzione deve essere tradotta in un linguaggio che il computer capisca, cioe' il "LINGUAGGIO MACCHINA". Un linguaggio al livello macchina e' detto anche "LINGUAGGIO di BASSO LIVELLO". Un linguaggio come il TI BASIC che assomiglia molto a frasi in inglese, viene chiamato anche "LINGUAGGIO ad ALTO LIVELLO". Il processo di interpretazione viene realizzato da un sistema di programmi, subroutines, e di dati, collocati nel computer dal fabbricante. Questo sistema e' conosciuto come "INTERPRETE BASIC".

L'interprete analizza ogni istruzione BASIC e converte queste istruzioni ad alto livello in un set di istruzioni a basso livello (o linguaggio macchina), che sono quelle che realmente fanno funzionare il computer. Non importa quale linguaggio di programmazione ad alto livello usiate, TUTTI devono essere tradotti nel codice macchina compreso dal computer. Il lavoro necessario per eseguire questo processo di interpretazione di come ogni istruzione viene eseguita e' la causa per cui i programmi scritti in BASIC o altri linguaggi ad alto livello girano piu' lentamente di un programma scritto in linguaggio macchina.

Il processo ASSEMBLY migliora enormemente la velocita' del programma. Quest'ultimo scritto in ASSEMBLY e' assemblato dall'assembler pressapoco come l'interprete tratta un programma in BASIC. Comunque con l'ASSEMBLY, il set risultante dei codici macchina puo' essere salvato su alcuni dispositivi per poterli riutilizzare. Poi, quando il programma viene fatto girare, non ci sara' alcun intermediario, come nel BASIC. Il codice macchina e' direttamente eseguibile dal computer, che cosi' eseguira' molto velocemente il programma caricato. Molti giochi, tipo arcade, e alcuni programmi applicativi scritti nel linguaggio ASSEMBLY del TMS9900, richiedono subroutines per rallentarli, perche ' troppo veloci.

Le istruzioni contenute in un programma scritto in linguaggio ad alto livello, come il TI BASIC, sono soltanto la sorgente del materiale dal quale poi viene generato il programma in linguaggio macchina. Nel linguaggio ASSEMBLY, il programma che voi codificate, viene chiamato " PROGRAMMA SORGENTE" o "CODICE SORGENTE" Quello assemblato dal vostro codice sorgente, e' chiamato "PROGRAMMA OGGETTO" o "CODICE OGGETTO". Poiche' essi sono sempre salvati su qualche dispositivo, potranno anche essere chiamati "FILE SORGENTE, o FILE OGGETTO". Dopo che il vostro programma in linguaggio ASSEMBLY e' stato assemblato, esso esiste in due forme: Il codice SORGENTE originale rimane immutato, mentre viene creato un nuovo file contenente il codice OGGETTO. (L'assembler linea per linea, come dice il suo nome, crea il codice macchina quando voi inserite la linea, cosi' il codice sorgente non e' salvato come file separato).

Oltre all'interpretazione, durante la sua esecuzione, in un linguaggio BASIC, vi sono anche altri inconvenienti. Il programma BASIC necessita di un vasto e sofisticato "Cast di supporto" hardware e software per farlo funzionare bene. Questo richiede che una parte delle risorse del computer siano dedicate a questo scopo, percio' non utilizzabili alle applicazioni del programma. Il TI BASIC e' progettato per essere facilmente leggibile dall'uomo. Sono usate parole comuni (Per gli Inglesi,pero'. N.d.T.), come PRINT, FOR, NEXT e DATA, e le istruzioni possono essere lette come una comune frase. (Sempre per gli Inglesi, N.d.T)

Questo modo di rappresentare le istruzioni e i dati per un essere umano, non e' necessariamente il modo piu' efficente per rappresentare la stessa informazione ad una macchina. Un programma in BASIC relativamente corto, molte volte produce piu' istruzioni macchina del numero di istruzioni BASIC che lo compongono.

Un'unica istruzione come:

```
10 INPUT X
```

Richiede molte istruzioni macchina per eseguire il compito stabilito. Il grado di complessita' coinvolto nell'esecuzione di una istruzione BASIC non e' visibile dal programmatore in BASIC.

Molti programmatori, ritengono che il BASIC incoraggi la gente a creare programmi non particolarmente efficenti, e che potrebbero essere trascurati, se paragonati allo stesso programma in ASSEMBLY.

Il LINGUAGGIO ASSEMBLY non e' un linguaggio macchina. E' ad un livello piu' alto, ma e' piu' vicino al linguaggio macchina del BASIC, percio' e' possibile scrivere programmi piu' efficenti in linguaggio ASSEMBLY che in BASIC.

Il CODICE OGGETTO occupa molto meno spazio in memoria, e su un dispositivo di memorizzazione, del CODICE SORGENTE. Una riduzione importante nel numero dei bytes richiesti per memorizzare ed eseguire un programma, lascia liberi piu' risorse e potenza in un computer, per la memorizzazione e la manipolazione dei dati.

Con il modulo dell'E/A e' possibile produrre il CODICE OGGETTO in formato COMPRESSO. Questo riduce anche ulteriormente le dimensioni del FILE OGGETTO per le esigenze di memorizzazione. Come si comprende subito, qualsiasi programma che viene scritto in maniera piu' corta e piu' semplice, girera' piu' velocemente, e usera' meno memoria.

Vi e' anche il rovescio della medaglia, naturalmente. In un linguaggio ad alto livello, come il TI BASIC, voi potete scrivere i programmi in semplici e corte istruzioni in Inglese, le quali possono affrontare alcuni compiti complessi in poche linee. E, modificare quelle istruzioni e' semplice poiche' voi non dovrete riassemblare il vostro programma prima di farlo girare. Potrete semplicemente battere RUN per vedere i risultati delle modifiche.

Con il liguaggio ASSEMBLY voi non avrete da codificare centinaia di istruzioni, ma dovrete codificare molte piu' dichiarazioni che in BASIC. Voi necessiterete di essere molto piu' specifici ed avere una maggiore capacita' di comprensione del vostro codice. Con il TI BASIC voi siete isolati da come lavora il computer. Non avete bisogno di sapere niente su come esso esegue i comandi che voi gli date, a voi bastera' solo conoscere la sintassi dei comandi dati per vedere dei risultati.

Con la programmazione in linguaggio ASSEMBLY, VOI siete al comando del computer, ed esso sara' a vostra completa disposizione. Pero' avrete bisogno di conoscere qualche altra cosa sui dettagli interni del computer.

Il valore dei particolari in un programma ASSEMBLY, sara' sempre maggiore di un'equivalente in BASIC. La maggior parte delle subroutine e dei programmi di supporto a cui eravate abituati in BASIC ed EX/BASIC qui' non ci sono. Voi dovrete progettare le vostre routines secondo le vostre necessita'.

Questo potrbbe sembrare un'inconveniente, o fastidio, ma e' realmente indicativo del grado di potenza che vi viene consegnato con il linguaggio ASSEMBLY. Voi avete l'opportunita' di usare la vostra creativita' progettando qualsiasi routines che serva al vostro uso e consumo. Potrete personalizzare molte delle funzioni che sono provviste per voi in BASIC.

Non abbiate timore dell'ASSEMBLY. All'inizio puo' sembrare opprimente, ma imparerete presto ad apprezzare tutto quello che esso puo' fare per voi,poiche' mette tutta la potenza del computer alla vostra portata. Vi sara' richiesto di piu' dalla vostra abilita' di programmatore, ma molto di piu' vi sara' consegnato. La rincompensa vi giungera' in fretta, sotto forma di accresciuta capacita', ed apprezzamento di come il computer esegue i compiti che voi gli date.

MANUALE DI RIFERIMENTO EDITOR/ASSEMBLER.

I seguenti riferimenti vi forniscono altre informazioni sul processo ASSEMBLY.

SEZIONE 1-1....Pag. 15
SEZIONE 15-1...Pag. 235

Leggere i seguenti termini sul GLOSSARIO.

ASSEMBLER
ASSEMBLING
ASSEMBLY LANGUAGE
COMPRESSED OBJECT CODE
MACHINE LANGUAGE
OBJECT CODE
SYNTAX
SYNTAX DEFINITION
TMS9900 Microprocessor

CAPITOLO SECONDO

NOTAZIONE BINARIA ED ESADECIMALE

Prima che possiate iniziare a programmare il vostro computer in un linguaggio di basso livello, come l'ASSEMBLY, e' importante capire come esso rappresenta ed elabora l'informazione.

Conoscendo come il vostro computer "pensa", voi potete orientare il vostro pensiero, e quindi l'approccio alla programmazione al livello della macchina sara' piu' facile. Attualmente il linguaggio macchina e' in forma BINARIA. Esso si riferisce al sistema di numerazione su cui sono basati i computers', cioe', il **CODICE BINARIO**, o a **BASE 2**. Qualche conoscenza di questo sistema e' necessaria per programmare in ASSEMBLY.

Un computer potrebbe essere ritenuto un vasto insieme di interruttori in miniatura, ciascuno dei quali puo' essere acceso o spento. Ognuno di questi interruttori puo' essere paragonato ad un BIT. La condizione di acceso o spento di un BIT puo' rappresentare molte cose: SI/NO, ALTO/BASSO, CALDO/FREDDO, o i valori di UNO e ZERO (che sono poi i valori che a noi interessano). Se un BIT e' ACCESO esso rappresenta UNO, se e' SPENTO, indica ZERO. Questo, naturalmente perche' solo due numeri sono usati (zero e uno), ed e' percio' che viene chiamato BINARIO.

Se voi considerate una serie di BITs, presi come un'unita', o come un valore espresso in forma binaria, grandi valori possono esere interpretati dagli stati "acceso/spento" di una serie di BITs.

Il sistema di numerazione che normalmente usiamo e' il **SISTEMA DECIMALE**, cioe' con **BASE 10**. Quando rappresentiamo un valore numerico decimale i simboli dei numeri rappresentano la potenza del "10". cosi' il numero "2139" puo'eessre scritto come segue:

$10^3 = 1000$	=	$2 * 1000$	=	2000
$10^2 = 100$	=	$1 * 100$	=	100
$10^1 = 10$	=	$3 * 10$	=	30
$10^0 = 1$	=	$9 * 1$	=	9

				2139

Le stesse regole vengono applicate ai sistemi numerici con base diversa da dieci.

Se voi state trattando con numeri significativi, avrete bisogno di piu' di un bit per poterlo fare, perche' un bit puo' solo indicare 1 o 0.

Un **BYTE** e' una serie di **OTTO BIT.SEDICI BIT** sono uguali a **DUE BYTEs**, o una **WORD**. Qui c'e' un BYTE rappresentato da otto numeri, ciascuno dei quali puo' solo essere un uno o uno zero.

00001101

In un sistema binario, ogni posizione rappresenta un'esponente della potenza di 2. Proprio come qualsiasi altro sistema numerico, la posizione degli zeri non ha

effetto sul valore dell'espressione, in tal modo per l'appunto esaminate i quattro bit piu' a destra. Questi sono anche detti BIT DI PESO MINORE o MENO SIGNIFICATIVI.

VALORE DEL BIT	1	1	0	1
VALORE DEL POSTO	otto	quattro	due	uno
$BASE^{esponente}$	2^3	2^2	2^1	2^0

Gli stati "acceso/spento" di questi bits rappresentano il valore "13", se presi come numero binario. Ecco come potete indicare questo valore nel sistema decimale:

```
1 * 8  =  8
1 * 4  =  4
0 * 2  =  0
1 * 1  =  1
-----------
         13
```

Che valore contiene questo BYTE ?

00000110

La risposta e' SEI. Sapete perche'?. E che ne dite di questo BYTE ?

00001111

Se avete detto "15", allora siete sulla strada giusta E di quest'altro BYTE?

00000001

Bene, uno e' sempre uno, anche in binario.

Mentre la notazione binaria applica prontamente gli stati di "acceso/spento" dei bits, scrivere tutti i valori nel formato binario puo' essere molto fastidioso, perche' e molto facile confondersi con tutti quegli zeri e uno, per cui normalmente si usa la "NOTAZIONE ESADECIMALE" o "BASE 16".

Ancora una volta ogni cifra in notazione ESADECIMALE rappresenta una potenza della base, che e' SEDICESIMA. Il valore dell'esponente in relazione alla posizione di ogni numero esadecimale dovrebbe essere:

VALORE DEL POSTO	4096	256	16	1
$BASE^{esponente}$	16^3	16^2	16^1	16^0

I numeri ESADECIMALI nel linguaggio ASSEMBLY del TI-99/4A vengono indicati con il simbolo "MAGGIORE DI" (">"), che precede il numero. Per es. il numero ">10" e' esadecimale, e corrisponde al decimale "16". Usando il modello esposto sopra, il numero a destra e' uguale a "0" percio' zero, mentre il numero a sinistra sara' "1" uguale a 16. Sommando i due numeri si ha alla fine 16+0 = 16, cioe', HEX >10.

Nel sistema decimale sono necessari 10 simboli (da 0 a 9) per indicare i valori. Con il sistema esadecimale (HEX) servono SEDICI simboli, cioe' da 1 a 9 come nel sistema decimale, e poi le prime SEI lettere dell'alfabeto per (A-F) indicare le rimanenti 6 cifre.

Cosi'A rappresenta il10, B rappresenta l'11, ecc. fino ad F per il 15

Il valore quindici nel codice binario occupa quattro cifre:1111. Nel codice HEX questo valore puo' essere indicato con una sola cifra: F. Cosi" un byte di dati viene indicato con due sole cifre, mentre in binario occorrono ben OTTO cifre, cioe' uni e zeri. Questo e' ovviamente un sistema piu' efficente di esprimere i valori. Ecco alcuni esempi.

DECIMALE	BINARIO	ESADECIMALE
1	00000001	>01
2	00000010	>02
3	00000011	>03
4	00000100	>04
5	00000101	>05
6	00000110	>06
7	00000111	>07
8	00001000	>08
9	00001001	>09
10	00001010	>0A
11	00001011	>0B
12	00001100	>0C
13	00001101	>0D
14	00001110	>0E
15	00001111	>0F
16	00010000	>10
32	00100000	>20
33	00100001	>21

Il valore piu' grande che puo' essere rappresentato con un byte (otto bit) e' il Binario 11111111, Hex >FF, o Decimale 255. Se voi esprimete una WORD (sedici bit) come espressione binaria, allora il valore piu' grande di una WORD sara' 1111111111111111, Hex >FFFF, o Decimale 65.535. Anche valori piu' grandi potranno essere impiegati, usando due o piu' WORD successive.

Per indicare se il valore di un numero e' positivo o negativo si usa il bit di sinistra del byte. Se questo bit e' OFF (spento) = ZERO, allora il valore rappresentato dai bit precedenti e' positivo. Se invece questo bit e' ON (acceso) = UNO, allora il valore e' negativo.

Il numero Binario indicato dai sedici bit 0111111111111111 sara' uguale a +32.767. Per i numeri piu' grandi di 32.767 il bit sinistro, o **BIT DI SEGNO**, dovra' essere usato.Per evitare conflitti i numeri piu' grandi di 32.767 vengono indicati come numeri **NEGATIVI** di **COMPLEMENTO A DUE**.

Il COMPLEMENTO A DUE, e' molto utile al computer per trattare con l'aritmetica binaria.

Supponete di voler calcolare "16 - 10". Il computer non puo' fare la sottrazione, ma dovra' invece compiere un'addizione in COMPLEMENTO A DUE. Il valore da essere sottratto viene convertito nel formato "Complemento a due", e addizionato al

primo valore. Cio' da lo stesso risultato della sottrazione. Poiche' questo e' logicamente lo stesso che negare il secondo valore, e addizionarlo.

Per affrontare il problema di cui sopra, esaminiamo i valori del bit, prima, durante e dopo il complemento aritmetico a due. Dal momento che usiamo numeri abbastanza piccoli, possiamo usare un byte (otto bits) per indicare ciascun totale.

	BINARIO	DECIMALE	HEX
VALORE 1	00010000	16	>10
VALORE 2	00001010	10	>0A

Primo, invertire tutti i bits 1 in 0, e i bits 0 in 1 per il valore 2

00001010 diventa 11110101

Notate che il primo bit a sinistra, adesso e' 1, dando a tutto il byte un valore negativo. Adesso addizionate uno al valore.

```
11110101+
       1=
---------
11110110
```

Il valore 2 e' ora nel formato COMPLEMENTO A DUE . Ora addiziona il valore 1 a questo.

```
00010000+
11110110=
---------
10000110
```

Trascurate il bit di sinistra, che e' stato riportato. I bit rimanenti sono 0000110 binario, cioe' uguale a SEI decimale. Infatti 16-10=6. Il computer usa il complemento a due per i numeri negativi e per qualsiasi valore piu' grande di 32.767. Questo e' importante per voi quando desiderate scrivere indirizzi maggiori di 32.767 nel formato decimale, in programmi che accedono a indirizzi specifici.

Una regola molto facile da seguire e' questa: PER QUALSIASI INDIRIZZO MAGGIORE DI 32.767, SOTTRARRE 65.536 DALL' INDIRIZZO.

Per esempio: per mettere il valore 79 all'indirizzo 33.008, usando la formula di cui sopra, si avra' 33.008 - 65536 = -32528 il codice dell'EX/BASIC dovrebbe assomigliare a questo:

```
10 CALL LOAD(-32.528,79)
```

Molto probabilmente sara' questo il maggior uso che farete della notazione in complemento a due, usando la notazione decimale per indicare gli indirizzi maggiori di 32.767.

Il complemento a due non si usa con la notazione esadecimale, infatti essa tratta i numeri piu' grandi di 32.767 senza nessun problema.

Esaminate questi esempi, e fate pratica esercitandovi a scrivere numeri in

otazione binaria e esadecimale. Scrivete ,per esempio , la vostra eta' 'in ogni otazione, oppure provate qualsiasi altro valore familiare. Ricordatevi che il rocedimento e' lo stesso per ogni sistema numerico, cambia solo la base della potenza he ogni numero rappresenta.

Se proprio non vi riesce vi sono dei calcolatori appositi che fanno la onversione da una base all'altra (ed esistono anche dei programmi per il TI-99/4A che anno la stessa cosa. N.d.T.)

Proprio come qualsiasi altra cosa, la notazione esadecimale diventa piu' facile an mano ci si esercita con essa. Il linguaggio ASSEMBLY del TMS9900 vi permettera' i indicare i valori in decimale, se voi lo desiderate, ma siccome la rappresentazione nterna del computer e' la notazione binaria, usando i numeri Hex si e' piu' vicini al uo modo di operare.

Ecco qui' sotto quattro problemi in aritmetica esadecimale, controllate se siete n grado di capirli.

```
1)   >6800+       2)   >7402+       3)   >D066+       4)   >0FAB+
     >029A=            >0EF0=            >110C=            >0A95=

     ------            ------            ------            ------
        ?                 ?                 ?                 ?
```

Ricordatevi che il simbolo (>) viene usato per indicare i numeri esadecimali. oiche' questo e' un sistema di numerazione in base 16, ci saranno poche differenze on l'aritmetica decimale. L'unica cosa che piu' confonde i principianti, sono le ettere da "A" ad "F", che qui' prendono il posto dei numeri da 10 a 15. Ecco un emplice esempio di addizione Hex e decimale.

ESADECIMALE	DECIMALE
>9 + >1 = >A	9 + 1 = 10
>A + >1 = >B	10 + 1 = 11
>B + >1 = >C	11 + 1 = 12
>C + >1 = >D	12 + 1 = 13
>D + >1 = >E	13 + 1 = 14
>E + >1 = >F	14 + 1 = 15
>F + >1 = >10	15 + 1 = 16

Notate che nel sistema decimale, dovete arrivare fino a nove prima di poter aggiungere una unita' alle decine, ed uno zero al posto delle unita'. Nel sistema HEX questo numero e' il QUINDICI (F), e vi e' il riporto di uno per indicare SEDICI, non DIECI, perche', ricordate, siamo in un sistema con base 16.

Controlliamo prima il problema 1: noterete che i numeri sono incollonati da destra a sinistra, proprio come una addizione decimale.

Ora saltate al problema 3. Nella colonna a destra vi e' la somma di 6 + C, cioe' 6 + 12 = 18, cosi' il riporto di uno viene messo sulla colonna subito alla sua sinistra. Il valore dell'uno riportato e' 16, perche' siamo in base 16, mentre il risultato della somma e' 2 (infatti 18 - 16 = 2).

Nel problema 4, la sottrazione si svolge in maniera simile.

Tornando indietro al problema 2, esaminate come si svolge la sottrazione. Nella seconda colonna da destra ,0 - >F un uno dovra' essere preso in prestito dalla colonna

alla sua sinistra. Questo valore preso a prestito e' SEDICI (ricordate sempre, siamo in base 16), percio' SEDICI meno QUINDICI = 1. Dopo aver preso un prestito dalla colonna seguente, questa diventa >3 - >E, per cui dobbiamo prendere un prestito ancora dalla colonna a sinistra. Addizionando SEDICI a TRE diventa DICIANNOVE, per cui 19 - 14 (>13 - >E) = 5. Nell'ultima colonna a sinistra, a causa del numero prestato, si ha SEI - ZERO = SEI.

Con un po' di pratica troverete che l'aritmetica esadecimale e' proprio facile come la decimale. Una volta che avrete preso l'abitudine ad usare i numeri HEX, non vi accorgerete piu' di nessuna differenza, ed usarli diventera' per voi naturale, ed e' meglio, perche' questo modo di rappresentare i BYTEs e le WORDs dei dati, vi dira' parecchie cose sullo stato dei bits, in poco spazio. Trovate un po' di tempo per fare pratica, addizionando e sottraendo numeri HEX>

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sui numeri BINARI ed ESADECIMALI.

APPENDICE
Dalla sezione 24-1,pag.393 alla sezione 24-1-4 pag.397.

CONTROLLA QUESTI TERMINI NEL GLOSSARIO

ASCII
BINARIO
BIT
BYTE
ESADECIMALE
COSTANTI INTERE ESADECIMALI
NYBBLE
COMPLEMENTO A DUE

CAPITOLO TERZO

INDIRIZZAMENTO

Ricordate per un momento il modello concettuale del computer, come un grande insieme di microinterruttori. Questo insieme e' analogo ad una mappa di vie cittadine,con palazzi,incroci, e intere comunita'. Per manipolare i bits e byte dei dati nel computer,voi dovete fare una mappa delle risorse del computer, indicando certe "comunita'", e assegnando gli indirizzi a queste aree e i singoli bytes.

Piu' avanti voi vedrete parecchi modi per specificare gli indirizzi nei programmi ASSEMBLY. Una maggiore attenzione ai passi del programma sara' necessaria per codificare e comprendere il movimento dei dati da un'area all'altra, o la manipolazione di un particolare bit, byte, o word.Il computer pretende che voi gli specifichiate l'esatta posizione dell'area a cui desiderate accedere, sia direttamente che indirettamente. Tutte le aree possibili del computer sono numerate in modo univoco, per identificare ognuna di esse. Il numero da cui il computer individua ogni byte, e' il suo **INDIRIZZO**.

Nella programmazione in linguaggio ASSEMBLY, viene usato un metodo chiamato "OFFSET" = "**BASE PIU SPOSTAMENTO** di ***INDIRIZZO***" per calcolare e numerare gli indirizzi interni. Dando un indirizzo di base conosciuto, avrete bisogno solo di calcolare il totale di OFFSET, o spostamento, necessario per arrivare all'indirizzo desiderato.

Quando contate i bytes, partite con zero come primo indirizzo del primo byte. ZERO e' il primo numero positivo del computer, percio', cominciate sempre a contare da ZERO, non da UNO. Consideriamo una area particolare della **VDP RAM** (Memoria ad accesso casuale del processore video), nella VDP RAM, il primo byte (byte zero) rappresenta la prima posizione utilizzabile dello schermo. In TI BASIC questa sarebbe nella RIGA 1 e COLONNA 1. Un byte rappresenta un carattere. Per esempio, se il byte zero della VDP RAM contiene il valore >41 (o decimale 65) la lettera "A" sara' visualizzata sullo schermo alla riga 1 e colonna 1. HEX >41 o decimale 65 e' il codice ASCII per la lettera "A".

E' mettendo i valori corretti in questa area di VDP RAM che i simboli ed i grafici sono fatti apparire sulo schermo. Molte delle funzioni che vorrete far fare al computer, richiederanno di mettere certi valori in questa specifica area del computer.

L'area attribuita al VDP RAM e' la "**TAVOLA IMMAGINE dello SCHERMO**" (T.I.S. da ora in avanti). Qui' vi sono 768 bytes che rappresentano ogni posizione dello schermo (24 righe * 32 colonne = 768). Gli indirizzi corrispondenti nella VDP RAM vanno da 0 a 767 decimale, (o da >0000 a >02FF Hex).

Allo scopo di rendere piu' facilmente leggibile il codice nel linguaggio ASSEMBLY, gli indirizzi non devono essere codificati con i loro valori numerici. Invece voi potete associare qualche nome significativo , o etichetta, con l'indirizzo. Quando voi vi riferite a questa etichetta, l'ASSEMBLER lo traduce per indicare questo indirizzo. Questo puo' essere fatto con l' istruzione EQUate (eguaglia).

Nel linguaggio ASSEMBLY del TMS9900, le etichette potranno essere lunghe al massimo SEI caratteri ASCII. Stabilire un'etichetta per il primo byte della TIS (tavola immagine dello schermo). Chiamare l'indirizzo in VDP RAM, dove la TIS inizia, per esempio, "SCRTAB".

Quella che segue e' un'istruzione ASSEMBLY che fa cio':

```
SCRTAB ECU >0000
```

Da ora in poi voi potete riferirvi al primo byte semplicemente come SCRTAB, o SCRTAB+0, dove +0 rappresenta uno spostamento del valore. Addizionando +0 a SCRTAB non si cambiera' il valore dell'indirizzo simbolico SCRTAB. Se voi scrivete un numero di 7 cifre, nela parte alta dello schermo, esso occupera' in VDP RAM, gli indirizzi seguenti: SCRTAB +0, SCRTAB +1, SCRTAB +2, SCRTAB +3, SCRTAB +4, SCRTAB +5, SCRTAB +6. Notate che lo spostamento dei valori, i primi sette byte della VDP RAM, partono da 0 fino a SEI. Per poter indrizzare l'ultima posizione dello schermo (riga 24, colonna 32)voi potete usare la notazione SCRTAB +767. Spesso un indirizzo necessario deve essere calcolato da qualche indirizzo di base, e qualche spostamento del valore.

Se il numero di cui sopra, era 5551234, allora i valori dei bytes in Hex dovrebbero essere:

NDIRIZZO SIMBOLICO ICHETTA BASE	SPOSTAMENTO VALORE	VALORE BYTE HEX	INDIRIZZO RELATIVO VDP RAM	CARATTERI RAPPRESENTATI DAL VALORE BYTE
SCRTAB	+0	>35	>0000	"5"
SCRTAB	+1	>35	>0001	"5"
SCRTAB	+2	>35	>0002	"5"
SCRTAB	+3	>31	>0003	"1"
SCRTAB	+4	>32	>0004	"2"
SCRTAB	+5	>33	>0005	"3"
SCRTAB	+6	>34	>0006	"4"

Se voi descrivete al computer un algoritmo per calcolare gli indirizzi, usando un'indirizzo di base,voi potete avere un programma che puo' "cercare la strada da solo", senza che voi dobbiate definire anticipatamente qualsiasi area interna necessaria. Mettete semplicemente: INDIRIZZO NUOVO = INDIRIZZO DI BASE + SPOSTAMENTO.

Alcuni diagrammi ed esempi nell'ASSEMBLY del TMS9900, dividono i singoli bytes allo scopo di mostrare lo stato o l'importanza di ciascuno degli otto bits che compongono il byte. Voi noterete che i bits vengono numerati cosi': 0,1,2,3,4,5,6,7. Se una WORD (due bytes o 16 bits) e' divisa allora i bits sono: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Questa e' un'altra applicazione dell'abitudine di contare sempre da zero.

Ricordatevi sempre, pero', che quando voi state indirizzando le aree del computer, i valori degli spostamenti contano i BYTEs, non BIT

La notazione SCRTAB +2 si riferisce ad un byte con l'indirizzo di DUE bytes piu' distante del byte SCRTAB. Ricordatevi che nella programmazione in linguaggio ASSEMBLY, zero e' il numero che rappresenta sempre il primo di una serie, quando viene

usato nel contesto di indirizzamento o posizione. Se voi avete fatto pratica in TI BASIC, vi ricorderete che il primo record di un FILE RELATIVO sara' sempre il record zero. Con l'istruzione DIM per definire una tavola, voi potete usare il parametro OPTION BASE 0. Questo stabilisce lo zero come primo indice di un ARRAY. Questi sono tipi similiari di BASE + SPOSTAMENTO.

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sull'INDIRIZZAMENTO.

APPENDICE
Dalla sezione 24-2 pag.398 alla sezione 24-2-2 pag.402

CONTROLLA QUESTI TERMINI NEL GLOSSARIO

Indirizzi
Modi di indirizamento
Consolle
CPU
Memoria
RAM
ROM
Indirizzamento simbolico della memoria
VDP RAM

Le mappe della memoria possono essere cercate sui manuali dell'E/A o M/M

Qui' sotto vi e' un altra vista dell'archittettura del TI-99/4A, che puo' aiutarvi nella visualizzazione delle varie locazioni di memoria.

```
                         TMS9900 CPU MEMORY

>0000        >2000       >4000      >6000      >8000      >A000
Console      Memory      Device      Command   CPU PAD     Memory
ROM          Expansion   Service    Module     Memory      Expansion
Operating    "Low" RAM   Routines    ROM/RAM   Mapping     "High" RAM
System                   ROM         Mini-        |        (24K bytes)
GPL                                  Memory       |
Interpreter                                       |
BASIC                                             |
Interpreter                                       |
                                                  |
                                                  |

                                   MEMORY MAPPED PORTS

>8000     >8300      >8400     >8800    >8C00    >9000   >9400   >9800  >9C00
768       256        Sound     VDP      VDP      Speech  Speech  GROM   GROM
Byte      Byte         |       Read     Write    Read    Write   Read   Write
Block     CPU PAD      |         |        |        |       |       |      |
                       |         |        |        |       |       |      |
                   -------   -----------------   -----------   ----------
                   TMS9919   TMS9918A Video      TMS5200       GROM
                   Sound     Display Processor   Speech        ----------
                   Chip                          Synthesizer   >0000
                               VDP RAM (16K)     -----------   3 Console
                             -----------------   Vocabulary    GROMs
                             >0000 Screen        ROM           containing
                                   Image         (32K bytes)   Monitor,
                             >0300 Sprite                      Operating
                                   Attribute                   System and
                             >0380 Color                       BASIC
                                   Table                       >4800
                             >0400 Sprite                      5 Command
                                   Descriptor                  Module
                             >0780 Sprite                      GROMs
                                   Motion
                             >0800 Pattern
                                   Generator
                             >1000 Free Space
                                   PABs/Buffers
                             >3500 Disk DSRs
```

CAPITOLO QUARTO

REGISTRI

Un registro e' una speciale WORD (16 bit, 2 bytes), indicata per memorizzare, che ha speciali potenze e responsabilita'. Ci sono 16 REGISTRI GENERALI "WORKSPACE", accessibili al programmatore ASSEMBLY del TMS9900. Questi registri di uso generale WORKSPACE sono numerati da 0 a 15. I registri sono il WORKSPACE della CPU (Unita' di elaborazione centrale), o SCRATCHPAD (Buffer di memoria CPU). Essi sono usati per l'aritmetica, indirizzamento, e manipolazione dei bits. Essi svolgono incarichi speciali che altre aree del computer non possono fare.

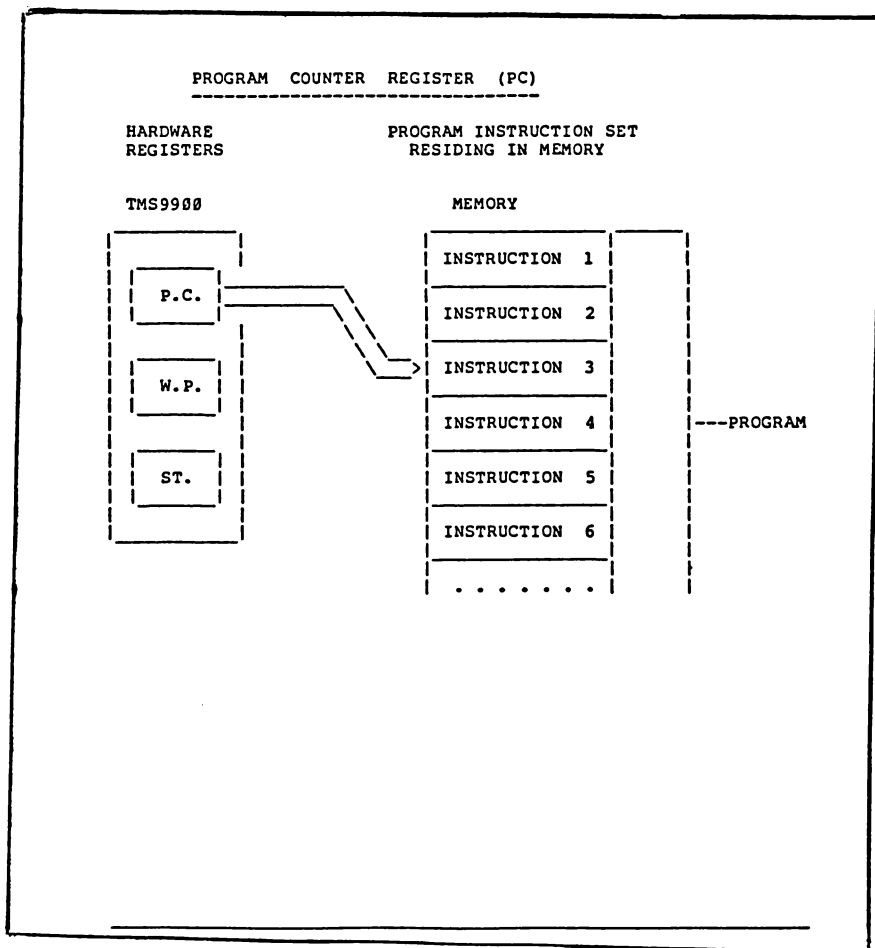
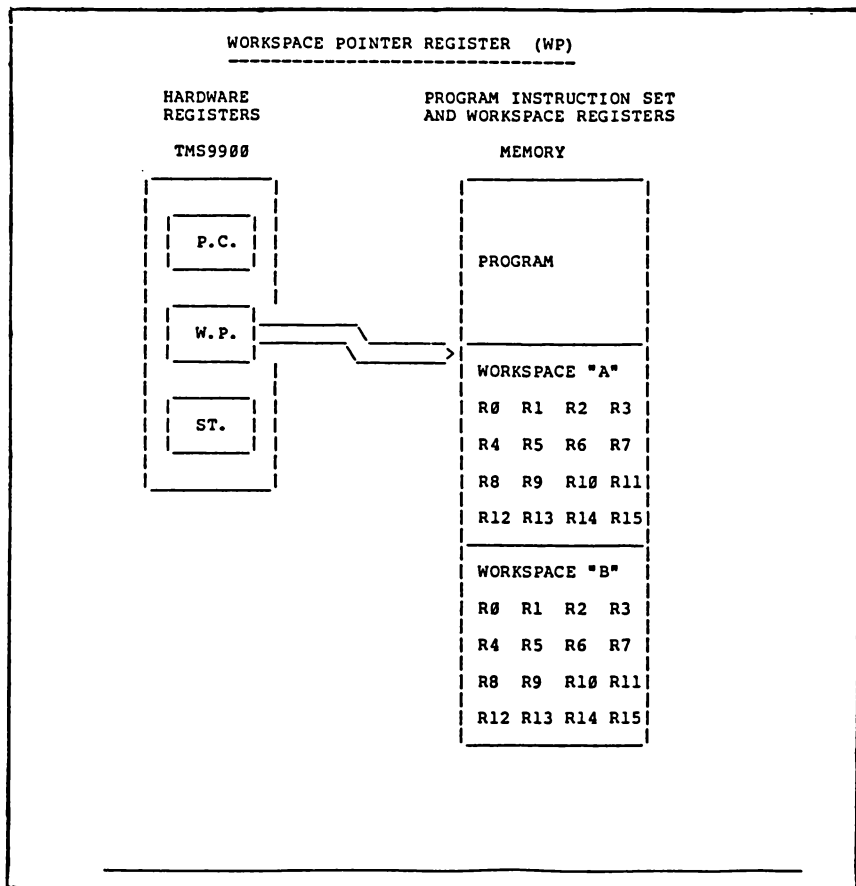
I registri generali WORKSPACE, occupano sempre un'area contigua di memoria, per un totale di >20 (32 decimale) bytes. (Ogni registro = 16 bit, o 2 byte. 16 * 2 = 32). Altri 32 blocchi di bytes di memoria possono essere designati dal vostro programma, per essere usati come registri generali WORKSPACE. Solo un set di 16 registri per volta puo' essere usato.

Se voi state usando il modulo dell'E/A, scegliete l'opzione "R" quando assemblate. Questo etichettera' automaticamente i 16 registri WORKSPACE come "R0...R15". Se voi state usando l'assemblatore LINEA PER LINEA della M/M, questi simboli sono predefiniti. Voi potete riferirvi a questi registri con i loro numeri (0, 1, 2, 3,...14, 15), se per voi e' piu' comodo.

Mentre i registri sono speciali nei loro usi e funzioni, la loro composizione e' identica ad altre aree di memoria. Ogni registro e' una serie di 16 bits il cui stato ON/OFF rappresenta alcuni valori nel formato binario. La notazione Hex per i contenuti di un registro e' data con quattro cifre. (16 bits, una cifra Hex per 4 bits). Esempio,>0020.

Oltre ai registri di uso generale WORKSPACE, vi sono TRE REGISTRI HARDWARE usati dal computer per il computer stesso e il programma mentre esso sta girando. Questi registri mantengono le indicazioni di indirizzi di subroutine s, dati, o altre risorse necessarie al vostro programma, la posizione della prossima istruzione da eseguire, lo stato risultante dell'ultima istruzione eseguita, e l'indirizzo iniziale dei registri generali WORKSPACE. I valori contenuti in questi registri hardware saranno importanti per voi quando farete i vostri programmi ASSEMBLY.

Il REGISTRO del CONTATORE DI PROGRAMMA (PC= Program Counter) mantiene l'indicazione di un set di istruzioni. I valori di questo registro sono usati in unione con altri indirizzi dati per localizzare o "PUNTARE" la prossima istruzione del vostro programma. Quando il vostro programma sta girando, tutte le informazioni contenute in esso sono memorizzate. L'indirizzo di memorizzazione che contiene il codice binario che rappresenta ogni istruzione del programma, viene gestito da questo registro. Come le istruzioni del programma sono eseguite, questo registro viene incrementato puntando all'indirizzo della prossima istruzione logica.

PROGRAM COUNTER REGISTER (PC)
HARDWARE
REGISTERS
PROGRAM INSTRUCTION SET
RESIDING IN MEMORY
TMS9900
MEMORY
P.C.
W.P.
ST.
INSTRUCTION 1
INSTRUCTION 2
INSTRUCTION 3
INSTRUCTION 4
INSTRUCTION 5
INSTRUCTION 6
---PROGRAM

Il REGISTRO PUNTATORE del WORKSPACE (WP= Workspace Pointer), contiene l'indirizzo, puntando all'indirizzo iniziale dei 16 registri generali del workspace. Molti gruppi di registri workspace possono essere definiti, e ogni gruppo puo' essere accessibile dalla manipolazione del programma degli indirizzi contenuti in questi registri. Questo puo' essere utile specialmente quando sottoprogrammi sono chiamati dal vostro programma, che necessita dei suoi propri registri.

Il REGISTRO di STATO (ST= Status Register), registra lo stato dell'ultima istruzione eseguita. I bit individuali sono fissati (1), o azzerrati (0), per indicare in certe condizioni come viene eseguita ogni istruzione. Il vostro codice del programma si riferira' a questo registro, sia direttamente, che indirettamente, molto spesso. Supponete di voler confrontare il valore X ed il valore Y. Immediatamente dopo che il computer ha eseguito l'istruzione di confronto, il registro di stato indichera' dallo stato dei suoi bit, alcuni relazioni:

REGISTRO di STATO

POSIZIONE BIT	SIGNIFICATO SE FISSATO A "1"
0	Logicamente maggiore di...
1	Aritmeticamente maggiore di...
2	Uguaglianza
3	Riporto
4	Traboccamento (Owerflov)
5	Parita' dispari
6	Operazione estesa
7 - 11	Non usati
12- 15	Maschera d'interruzione

Il registro di stato puo' permettervi di conoscere se due valori sono uguali, o dare evidenza a molte altre condizioni.Le operazioni aritmetichetrattano tutti i bytes come rappresentanti un valore aritmetico.

Le operazioni logiche trattano i bytes come rappresentanti una serie di bit.
Varie istruzioni disponibili per voi nel linguaggio ASSEMBLY, vi permetteranno di istruire il computer nel modo in cui desiderate che i contenuti di un byte vengano valutati. L'uso di questo registro verra' ripreso nei capitoli che trattano questo codice.

Alcune istruzioni nel linguaggio ASSEMBLY si applicano solo ai registri, o sono richieste per coinvolgere almeno un registro. Il trasferimento dei dati, verso e dai sottoprogrammi speciali, e' effettuato attraverso certi registri, come e' il movimento dei dati verso e da speciali aree del computer. Calcoli di indirizzi speciali e indici, sono possibili con i registri.

I tre registri hardware (PC, WP, ST), e i 16 registri generali del workspace eseguono la maggioranza dei calcoli numerici, e assistono in quasi tutte le altre fasi della gestione e manipolazione dei dati.

Parecchi fabbricanti di computers, e di linguaggi ASSEMBLY, offrono un numero differente di registri di lavoro (Workspace). Alcuni ne hanno solo 3, altri 8, o 16, o 32. Voi non avrete mai bisogno di scrivere programmi che utilizzino tutti i 16 registri del TMS9900, ma il potenziale esiste. I registri sono un componente del microprocessore, e sono progettati nel set di istruzioni del processore. Per lui e' molto piu' veloce ed efficente operare sui suoi registri, che usare un'indirizzo in memoria piu' distante.

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sui REGISTRI.

Dalla sezione 3-1 pag. 39 alla sezione 3-1-3 pag. 40.

LEGGERE QUESTI TERMINI SUL GLOSSARIO.

BIT Aritmeticamente maggiore di...
BIT di riporto (Carry)
BIT di uguaglianza
BIT logicamente maggiore di...
BIT di parita' dispari
BIT di traboccamento (Owerflow)
Registro di contatore programma (PC)
Registro
Registro di stato (ST)
Workspace
Registro puntatore del Workspace (WP).

IMPORTANTE

Prima di procedere alla prossima sezione di questo libro, vai all'E/A, alla fine del capitolo cinque. Leggi inoltre le pagine indicate sopra, e i termini del glossario. Poi rileggi il capitolo, e poi ancora il manuale E/A, se necessario.

CAPITOLO QUINTO

CODIFICAZIONE

Per mettere in pratica la teoria studiata fin'ora, voi dovrete imparare a codificare un programma in ASSEMBLY. Ma per prima cosa, prendete un po' di tempo per leggere il manuale di riferimento dell'E/A nei punti indicati "alla fine di questo capitolo. Quando avrete fatto cio', osservate questo semplice programma in TI BASIC. Esaminate ogni istruzione ricordando quello che e' stato discusso nelle sezioni precedenti.

```
10 CALL CLEAR
20 AMTX=10
30 AMTY=33
40 AMTY=AMTX+AMTY
50 PRINT AMTY
60 END
```

DISCUSSIONE DELLE VARIE ISTRUZIONI.

10 CALL CLEAR. Questa istruzione pulira' lo schermo da qualsiasi carattere in quel momento visualizzato. Il programma in TI BASIC e' "chiamato" da un sottoprogramma residente nel TI BASIC con il nome "CLEAR". Questa routine fara' tutto quello che e' necessario per pulire lo schermo, e poi ritornare il controlo alla prossima istruzione, cioe' la linea...

20 AMTX=10. Che riserva uno spazio di memoria assegnato all'etichetta "AMTX". Poi inizializza AMTX con il valore aritmetico di 10.

30 AMTY=33. Mette da parte un altro spazio di memoria. Da' a questo spazio il nome "AMTY", e inizializza AMTY con il valore aritmetico di 33.

40 AMTY=AMTX+AMTY. Addiziona il valore dell'indirizzo AMTX a quello di AMTY, e mette il totale nell'indirizzo di AMTY.

50 PRINT AMTY. Visualizza il valore memorizzato all'indirizzo "AMTY",con i simboli ASCII. La posizione sullo schermo dove verra' visualizzato questo risultato, e' gia' stata predeterminata dal TI BASIC come l'angolo inferiore sinistro. (Posizione di DEFAULT)

60 END. Ritorna il controllo del computer al sistema operativo.

Come voi potete vedere, ci sono molte cose che il computer deve fare per eseguire ogni istruzione in TI BASIC. Pochi dettagli di questi compiti vengono visti da chi opera. L'interprete BASIC, ed il resto del " Gruppo di supporto" si prendono cura di tutto questo. In un programma ASSEMBLY, voi dovrete essere molto piu' precisi nell'istruire il computer sui suoi incarichi, e su come eseguirli.

Provate a codificare questo programma ASSEMBLY, per far eseguire la stessa cosa del programma BASIC di cui sopra. Ogni istruzione ASSEMBLY e' costituita da TRE parti

piu' importanti, chiamati CAMPI, cioe': Un' "ETICHETTA", un " CODICE OPERATIVO " (OP-CODE), o istruzione, e UNO o DUE OPERANDI.

Ricordate l'esempio precedente di istruzione ASSEMBLY ?:

SCRTAB EQU >0000

SCRTAB E' L'ETICHETTA. L'etichetta parte dalla prima posizione a sinistra della linea, e NON puo' essere lunga piu' di SEI caratteri (due con l'assemblatore LINEA per LINEA), e il primo carattere DEVE essere alfabetico.

L'etichetta e' un'opzione e non e' sempre necessaria. Il codice operativo (OP-CODE) e' separato dall'etichetta da almeno uno spazio. L'OP-CODE e' l'attuale istruzione del programma. Il campo operando e' separato a sua volta dall'OP-CODE di almeno uno spazio. Ci potranno essere, come gia' detto, UNO o DUE operandi. Se sono due, essi dovranno essere separati da una virgola.

Il campo operando identifica il registro o un'altro indirizzo su cui l'istruzione (OP-CODE) sta operando.

Quando sono necessari dei commenti, essi dovranno iniziare almeno uno spazio a destra dopo l'ultimo operando. Iniziando una nuova linea con un'asterisco (*), tutta la linea potra' essere dedicata ai commenti. (L'asterisco ha la stessa funzione della REM in TI BASIC)

Ecco qui' sotto un piccolo programma in ASSEMBLY del TMS9900, Scritto con il package dell'E/A. La "riga" numerata da 1 fino a 50 indica le colonne, ed e' inclusa qui' per illustrare le posizioni relative di ogni campo.

NOTA DEL CORRETTORE DELLA TRADUZIONE.

Il listato del programma di cui sopra viene stampato a parte,ed allegato, in maniera da poterlo avere a portata di mano quando si controlla la spiegazione del programma stesso, cio' evita noiosi ritorni avanti e indietro con le pagine per controllare listato e spiegazioni.

Non vi allarmate dell'alto numero di istruzioni. I linguaggi ASSEMBLY sono sempre piu' "Verbosi" dei linguaggi ad alto livello, come il BASIC. Ogni linea sara' analizzata e spiegata con molti utili comandi lungo la via. Alcune istruzioni in un programma in linguaggio ASSEMBLY, sono richieste di ingresso, e mostrano programma dopo programma.

Una volta che avrete scritto diversi programmi in ASSEMBLY, esse diventeranno superflue. Come voi imparerete a progettare le vostre subroutines, esse potranno essere usate in qualsiasi altro programma, senza doverle rifare ogni volta. Nonostante la loro apparente (?) complessita', tutte le operazioni, sono, ma con variazioni costanti, sul principio dei BITs e BYTEs, INDIRIZZAMENTI, REGISTRI, ARITMETICA BINARIA, e cosi via. (E non mi sembra poco...N.d.T.)

Il programma e' stato scritto in maniera semplice, per dimostrare le funzioni specifiche con cui siete familiari. Lo stesso programma potrebbe essere scritto in maniera piu' breve e "scorrevole", ma, poi a noi interessa un programma per fare 10+33 ?!

Ecco qui' la spiegazione di ogni linea ASSEMBLY.

LINEA 01 DEF START

DEFine, (DEFinisci) e' una DIRETTIVA ASSEMBLER. Una DIRETTIVA, e' una istruzione dell'ASSEMBLER, che e' necessaria per il particolare ASSEMBLY del programma. Non c'e' nessun contrasto con l'esecuzione logica del programma. La DIRETTIVA DEF ha l'effetto di mettere il nome dato al programma ("START"), dentro un'area del computer conosciuta come TAVOLA REF/DEF. Qui' vengono conservati i nomi di tutti i programmi che sono correntemente in memoria.

La direttiva DEF assicura che quando il vostro programma e' caricato, il suo nome sara' aggiunto alla TAVOLA REF/DEF. Quando voi fate girare il programma, questo e' il punto in cui il software RUN vede il suo nome.

START e' l'indirizzo simbolico del punto dove il vostro programma inizia l'esecuzione. La direttiva DEF deve precedere l'etichetta che esso definisce. La pratica piu' comune e' semplicemente quella di metterla come prima istruzione del vostro programma .

L'etichetta usata puo' essere qualsiasi nome di etichetta valido, "START" e' il nome usato nel programma esempio.

LINEA 02 REF VSBW,VMBW

La direttiva REF dice all'assembler che voi intendete usare alcuni programmi speciali residenti. Essa accede anche alla TAVOLA REF/DEF. Questa direttiva assicura che quando il vostro programma e' caricato, queste routines saranno disponibili per esso. VSBW sara' reso uguale all'indirizzo della routine in VDP RAM per scrivere un singolo byte (Vdp Single Byte Write). VMBW sara' reso uguale all'indirizzo della routine in VDP RAM per scrivere piu' bytes (Vdp Multiple Byte Write).

Queste routines sono usate per visualizzare sullo schermo grafici e caratteri. Esse sono proprio due membri del "gruppo di supporto" del linguaggio ASSEMBLY del TMS9900.

LINEA 03 STATUS EQU>837C

Questa e' l'istruzione EQUate (eguaglia). Essa e' anche una direttiva. L'indirizzo e' quello dello "STATUS BYTE" (Byte di stato). Voi vi riferirete allo STATUS BYTE nel vostro programma, riferendovi all'attuale indirizzo dell'etichetta simbolica "STATUS".

La direttiva EQUate, associa l'etichetta data con l'indirizzo >837C. L'indirizzo attuale unito al simbolo STATUS viene caricato durante il processo ASSEMBLY dentro un'area chiamata "TAVOLA dei SIMBOLI". L'assembler usa la TAVOLA dei SIMBOLI per trovare l'indirizzo proposto ogni volta che voi userete una etichetta simbolica come STATUS. Voi dovete definire queste relazioni tra indirizzi di macchina e i loro nomi simbolici, con la direttiva EQUate.

04 SAVRTN DATA >0000
05 AMTX DATA >000A
06 AMTY DATA >0021
07 DECTEN DATA >000A

Queste linee usano le direttive DATA,. Questa direttiva e' usata per inizializzare una WORD (parola) (16 bit=2bytes) di memoria per qualche valore. Se una etichetta e' inclusa, quest'ultima viene associata con l'indirizzo iniziale della

WORD.

L'etichetta rappresenta un'indirizzo simbolico. L'operando contiene il valore della word che sta' per essere fissata. Il valore puo' essere scritto in notazione decimale o esadecimale. Usando gli indirizzi simbolici ogni volta che e' possibile, voi non dovrete tenere il conto dei valori dell'indirizzo.

Le etichette che voi ideate avrano sempre un valore mnemonico (aiutando la vostra memoria). Il vostro programma sara' piu' leggibile e comprensibile, se le etichette scelte saranno in relazione a quello che esse definiscono.

AMTX DATA >000A Approssimativamente uguale a: AMTX=10
AMTY DATA >0021 Approssimativamente uguale a: AMTY=33

Una direttiva simile e' la direttiva BYTE. L'istruzione MYBYTE BYTE >04, inizializza un byte (8 bits) di memoria. Gli effetti dei DATA e BYTE sono similari, l'unica differenza e' il numero dei bits che sono inizializzati (8 o 16). L'assembler "Linea per linea" non riconosce la direttiva BYTE.

09 PNTANS BSS 2
10 WSPREG BSS >20

Le linee 9 e 10 usano la direttiva "BSS", BLOCK STARTING with SIMBOL. Questo riserva i blocchi di memoria senza qualsiasi inizializzazione. Queste aree saranno usate come workspace dal programma, e saranno parte del programma stesso. Nella linea 9, sono stati riservati due bytes di memoria. Essi saranno attribuiti a "PNTANS". La linea 10 mette da parte 32 (>20) bytes chiamati "VSPREG".

Il primo quarto di questo programma e' stato descritto, e nessuna istruzione e' ancora stata eseguita. Tutto quello che e' stato fatto finora puo' essere considerato una preparazione. Quando codificate le istruzioni in TI BASIC, voi potete definire le variabili nella stessa istruzione in cui sono usate per la prima volta. Non e' cosi nel linguaggio ASSEMBLY. Voi dovete definire tutte le etichette e le aree del workspace, prima che possiate riferirvi a loro nelle istruzioni del programma

LINEA 11 START MOV R11,@SAVRTN

Qui' c'e' l'etichetta "START" che e' stata definita alla linea 1. Il primo scopo e' quello di salvare l'indirizzo di entrata del programma. Il registro R11 e' il registro che ha il compito di collegamento (Indirizzamento) generale del computer. Quando il vostro programma ASSEMBLY inizia l'esecuzione, l'indirizzo al quale esso dovrebbe ritornare una volta fatto, e' in R11. Questo indirizzo e' essenziale per il completamento con successo del programma stesso. Voi dovrete salvare questo indirizzo, poiche' R11 sara' usato altrove nel programma. La word di memoria chiamata SAVTRN era stata messa da parte proprio per questo scopo. MOVe (Muovi) il valore in R11 alla posizione di memorizzazione.

L'indirizzo usato e' un indirizzo simbolico, che e' rappresentato in una istruzione MOVe dal simbolo § "AT". L'operazione MOV copia una word (16 bits, 2 bytes) di un registro, o altro indirizzo in memoria a un'altro indirizzo o registro. Il registro che contiene la word da copiare rimane invariato, mentre la posizione di memorizzazione ricevente diventa la sua copia.

Supponiamo che R11 contenga il valore indirizzo >3238. Prima dell'istruzione

MOV:

R11 >3238 @ SAVRTN >0000

Dopo MOV, si ha:

R11 >3238 @ SAVRTN >3238

Notate che si e' usata una intera word (16 bits, 2 bytes). Supponiamo di dover spostare solo un bytes (8 bits) alla volta. Allora si dovra' usare l'istruzione **MOVB** (Muovi un byte). In tutto il linguaggio ASSEMBLY del TMS9900 voi troverete istruzioni parallele, il cui indirizzo e' una word (16 bits) o un byte (8 bits) alla volta. Se voi usate l'istruzione byte con un registro, o un' altro indirizzo word, l'istruzione usera' sempre il byte di sinistra, o byte piu' importante. Per esempio:

MOVB R3,R4

Supponiamo che i contenuti dei registri usati, prima che l'istruzione sia eseguita, siano:

R3 >104C R4 >0011

Allora si avra', dopo MOVB:

R3 >104C R4 >1011

Notate come il byte destro (Byte meno importante) rimanga inalterato, in ciascun registro, dalla istruzione MOVB.

LINEA 12 LWPI WSPREG

Ora voi LOAD WORKSPACE POINTER IMMEDIATE "LWPI" (Carica immediatamente il puntatore dello spazio di lavoro). Voi avrete bisogno di stabilire un'area alternativa dei registri workspace per l'uso di routine speciali che sono necessarie. Esse possono avere il loro proprio gruppo di registri generali workspace. L'effetto di questa istruzione e' quello di puntare all'indirizzo del blocco di memoria che era definito alla linea 10. Questa e' una istruzione tipica di cui avrete bisogno in qualsiasi programma autonomo nel linguaggio ASSEMBLY del TMS9900.

LINEA 13 BL@CLEAR

Invece di chiamare una routine residente per pulire lo schermo, questo programma ha una sua routine codificata. L'istruzione "BRANCH and LINK" (BL), (Diramati e allacciati) e' all'incirca equivalente al GOSUB con RETURN del BASIC. Il controllo e' passato da questo punto del programma, all'indirizzo CLEAR, e ancora una volta l'indirizzo di ritorno (indirizzo della prossima istruzione da eseguire) e' caricato nel registro R11. Quando codificate le istruzioni in BASIC, e dovete fare un GOTO ai numeri di linea, qualche volta risequenzando, potrebbe essere disastroso, se avete un numero di linea non ancora definito. Una delle finezze nell'uso di etichette, e che i numeri di linea non hanno effetto sul programma logico.

CLEAR si riferisce all'indirizzo iniziale della subroutine CLEAR, e non importa su quale linea e' il numero. R11 ora contiene l'indirizzo della linea 14, che e'

quella dove voi dovete ritornare quando e' finita la routine CLEAR.
L'esecuzione del programma si trasferisce ora alla ...

LINEA 34 CLEAR CLR R0
LINEA 35   CLR R1

"CLEAR" e l'etichetta alla quale voi avete istruito il computer per diramarsi. Il primo passo della routine CLEAR e' di fissare a zero tutti i bit dei registri 0 e 1. l'istruzione CLEAR pulisce (cioe' , mette a zero tutti i bit) una word di memoria, o un registro, alla volta. R0 e R1 adesso contengono:

R0 >0000   R1 >0000

LINEA 36 LOOP BLWP @VSBW

Ora voi potete iniziare a "Giocherellare" con il "Chip" del VDP (Video Display Processor). L'etichetta "LOOP" sara' usata per costruire un semplice ciclo, molto simile al ciclo FOR-NEXT del TI BASIC. Un ciclo (Loop) vi permette di eseguire un'istruzione molte volte.

l'istruzione BLWP, e' simile ad "Diramati e allacciati" (Branch and Link), salvo che questa volta voi volete che il registro del puntatore workspace (WP) punti ai registri workspace alternativi stabiliti alla linea 11.

Questo richiede l'uso della VSBW e di altre routines residenti. BLWP vuol dire, Branch and Link Workspace Pointer (Diramati ed allacciati al puntatore workspace). L'indirizzo a cui l'esecuzione del programma si dirama, e' l'indirizzo della routine in VDP RAM Single Byte Write "VSBW" (Scrivi un solo byte). Voi potete passare i valori a questo programma attraverso R0 e R1

In R0 voi mettete l'indirizzo di destinazione in VDP RAM, nel quale volete scrivere, e mettete il byte dei dati nel byte sinistro (quello piu' importante) di R1.

Generalmente R0 e R1 contengono tutti zeri. Ricordate che l'indirizzo zero in VDP RAM corrisponde alla riga 1 e colonna 1 dello schermo e R0 in VDP RAM. La routine VSBW ha scritto a questo indirizzo il byte sinistro, quello piu' importante,(8 bits) di R1 (tutti zeri). Che cosa verra' visualizzato a riga 1 e colonna 1 ? Niente Poi se voi aggiungete 1 al valore in R0 e ripetete questo passo, l'indirizzo 01 in VDP RAM (riga 1 e colonna 2) verra' pulito. Ci sono 768 posizioni dello schermo da pulire, e questi indirizzi in VDP RAM vanno da 0 a 767 (in decimale), o da >0000 a >02FF.

Cosi' ciclo dopo ciclo, attraverso questi passi, finche' R0 eù stato incrementato fino al valore di 767, e l'intero schermo e' pulito. La prossima istruzione controlla R0 per questo vaore.

LINEA 37 CI R0,767
LINEA 38 JEQ CLEARX

La linea 37 e' un'istruzione di COMPARAZIONE IMMEDIATA (CI). Questa e' usata per confrontare il valore di un registro con un valore conosciuto. Come risultato, i bits del registro di stato, sono influiti, ed esaminati dalla prossima linea. In piu' , per confrontare i registri per conoscerne i valori, ci sono istruzioni di comparazioni di word per word, e confronti di byte per byte.

L'istruzione Jump if EQual (JEQ), (Salta se uguale) completa il confronto, dirigendo alcune azioni basate sul risultato. Questa istruzione controlla il bit di uguaglianza del registro di stato (ST) e, se fissato a uno, trasferisce il controllo all'etichetta CLEARX.

Le istruzioni JUMP (salto) sono simili alle istruzioni GOTO del BASIC. Gli indirizzi che esse usano devono stare dentro a 256 bytes dell'istruzione stessa. Se la differenza è maggiore, il messaggio d'errore "OUT OF RANGE" apparirà durante l'assembly. Le istruzioni JUMP non necessitano del prefisso "@" negli indirizzi simbolici. La prima volta che si entra in questo ciclo, la condizione di uguaglianza non sarà vera, e l'istruzione JEQ della linea 38 non avrà effetto sull'esecuzione del programma.

LINEA 39 INC R0

INCrementare il valore di R0 con il valore binario 1. Ricordate che questo valore sarà usato come un indirizzo in VDP RAM. Ogni volta voi, attraverso questo, incrementerete quell'indirizzo di 1. L'istruzione "INC" incrementerà (aggiungendo un 1 binario) il registro, o la word in memoria specificata nell'operando. C'è anche una istruzione "INCT" che incrementerà di 2.

```
LINEA 40            JMP  LOOP
LINEA 41   CLEARX   B    *R11
```

La linea 40 è un salto incondizionato all'etichetta LOOP, che completa il ciclo descritto sopra. Quando il valore 767 è stato raggiunto in R0, l'esecuzione del programma si trasferisce alla linea 41, CLEARX. Questa istruzione è una diramazione incondizionata (Branch) "B" (simile a GOTO del BASIC), all'indirizzo in R11. L'uso dell'asterisco (*) immediatamente prima del registro nominato, indica che il valore in R11 può essere usato come un'indirizo. L'istruzione "BL" della linea 13, mette l'indirizzo della linea 14 in R11, prima che la subroutine CLEAR venga eseguita. Adesso voi state istruendo il computer perchè si dirami all'indirizzo in R11, che è l'indirizzo della linea 14.

LINEA 14 ADDUP A @AMTX,@AMTY

L'etichetta ADDUP vi aiuta a ricordare è stato fatto. I contenuti della word (16 bits, 2 byte) all'indirizzo simbolico AMTX vengono aggiunti al valore all'indirizzo AMTY. Entrambi gli indirizzi richiedono il prefisso AT (@) per questo passo.

Prima dell'indirizzo:	@AMTX	AMTY
	>000A	>0021
Addiziona		>000A
Dopo l'addizione		>002B

Adesso il risultato (>002B) è in AMTY. Ma il valore in AMTY è un valore binario, non nel codice ASCII corretto per visualizzare i caratteri "43" (equivalente dell'esadecimale >002B) sullo schermo. Avrete quindi bisogno di visualizzare un >34

(codice ASCII per il simbolo "4") in una posizione dello schermo, e un >33, (ASCII per "3") nella prossima posizione dello schermo, allo scopo di poter visualizzare il numero "43".

La prossima serie di istruzioni convertirà il risultato nel formato visualizzabile.

LINEA 15 MOV @AMTY,R5
LINEA 16 CLR R4

I registri 4 e 5 saranno usati per le operazioni aritmetiche necessarie. Il risultato (ancora in formato binario) è mosso nel R5, e R4 è pulito.

LINEA 17 DIV @DECTEN,R4

L'istruzione DIVide (Dividi) fa proprio questo; divide. DIV usa due registri successivi, in questo caso, R4 e R5 . Voi avrete bisogno solo di specificare il registro R4 nel secondo operando, poiché l'uso del prossimo registro disponibile è implicito. Il primo operando, DECTEN è il divisore. Questa istruzione divide il valore in R5 dal valore in DECTEN, e mette il risultato in R4, e il resto eventuale in R5. Dopo l'istruzione DIV:

@DECTEN	R4	R5
>000A	>0000	>002B
	dopo la divisione	
>000A	>0004	>0003

Nel sistema decimale ciò sarebbe uguale a 43 diviso 10, cioè : 4 con il resto di 3.

LINEA 18 MASKUP A @HEX30,R5

Questo addiziona il valore a HEX30 al valore in R5, e mette il risultato in R5. Prima che la linea 18 venga eseguita, R5 contiene >0003 (il binario 3). Il codice ASCII per visualizzare "3" è >33. La differenza tra questo e il risultato è di >30 (>33 meno >03 = >30). Questo >30 o "MASK" HEX30, deve essere addizionata al valore binario per renderlo uguale al proprio numero in codice ASCII. Prima dell'addizione:

@HEX30	R5
>0030	>0003

Dopo l'addizione:

R5 ora contiene il codice ASCII di "3". Questa è la prima cifra del vostro risultato visualizzabile.

LINEA 19 MOV R5,@PNTANS

Salvare questa parte del risultato nell'area che era stata riservata alla linea

9, PNTANS. Questa è una word che muove 16 bits. Prima di MOV:

```
R5  >0033          @PNTANS  >0000
```

Dopo MOV

```
R5  >0033          @PNTANS  >0033
```

```
LINEA 20 MOV  R4,R5
LINEA 21 CLR  R4
LINEA 22 DIV  @DECTEN,R4
LINEA 23 A    @HEX30,R5
```

Le linee da 20 a 23 ripetono il processo per la seconda cifra del risultato. Quando 43 era diviso per 10 alla linea 17, il risultato 4 era messo in R4. Poi esso era nuovamente diviso per 10. Per fare questo è necessario metterlo in R5, e fissare a zero R4. Quattro diviso 10 da 0 come risultato con il resto di 4. Il resto (4) è messo in R5, a cui viene aggiunto un MASK di >30. In questo tipo di conversione logica, voi state operando su R5. R5 contiene ora >0034 (L'ASCII di "4").

```
LINEA 24 SLA   R5,8
LINEA 25 MOVB  R5,@PNTANS
```

Ora voi avete la parte "4" del "43" che desideravate visualizzare. Il prossimo passo da fare è quello di muoverlo al PNTANS, e avvicinarlo al "3". Poichè voi non volete distruggere il "3" che è ora in PNTANS, sarà meglio muovere un byte invece che una parola. Ricordate che le istruzioni per i byte operano sempre sul byte sinistro (Quello più significativo). R5 contiene >0034, il che vuol dire che il valore da muovere è nel byte "sbagliato". (Ci sono molti modi per fare tutto questo e il metodo usato quì è congegnato per illustrare l'uso dell'istruzione). Una maniera è mostrata alla linea 24. Questa è un'istruzione SHIFT (Sposta), ed è una delle cose speciali che solo i registri possono fare. La particolare istruzione usata è quella di "SHIFT LEFT ARITMETIC" (Spostamento aritmetico sinistro) SLA. La linea 24 specifica che i bits nel registro 5 devono essere spostati di 8 posizioni a sinistra, e che la parte destra del registro deve essere riempito con zeri. Qui sotto sono indicati i contenuti di R5 diviso nei suoi bits individuali, prima e dopo che l'istruzione SLA venga eseguita:

```
Prima di SLA:        REG. 5 -Binario         HEX

BIT       0 1 2 3 4 5 6 7 8 9 A B C D E F
VALORE    0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0     >0034

            <------Sposta di 8 posizioni

Dopo SLA

          0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0     >3400

Prima di MOVB:      R5  >3400       @PNTANS  >0033

Dopo MOVB           R5  >3400       @PNTANS  >3433

LINEA 26   PUTUP  LI   R0,738
LINEA 27          LI   R1,PNTANS
LINEA 28          LI   R2,2
LINEA 29          BLWP @VMBW
```

Le linee da 26 a 29 usano le subroutine in VDP RAM (VMBW), per visualizzare il risultato finale. Le linee 26, 27, e 28 usano l'istruzione "LOAD IMMEDIATE" LI (Carica immediatamente). LI è usato per mettere i valori nei registri. Come l'istruzione COMPARA IMMEDIATAMENTE (CI), LI è usato con valori specifici. Il primo operando indica un registro, il secondo, il valore. R0 è caricato con l'indirizzo in VDP RAM della posizione desiderata sullo schermo. R1 è caricato con l'indirizzo, (simbolico o reale), del dato da muovere (non il dato stesso, ma l'indirizzo iniziale dove il dato è da cercare). In R2 è caricata la lunghezza, in bytes, del dato da muovere. La diramazione alla subroutine mette >3433, trovato all'indirizzo PNTANS, in VDP RAM indirizzo 738 (Angolo basso a sinistra, dello schermo). La VDP RAM all'indirizzo 738, contiene >34 (codice ASCII per 4), e la VDP RAM all'indirizzo 739 contiene >33 (codice ASCII per 3).

```
LINEA 30  EOJ  MOV   @SAVRTN,R11
LINEA 31       CLR   @STATUS
LINEA 32       DECT  R11
LINEA 33       RT
```

Queste linee completano il programma. Nella linea 42, la direttiva END, è una richiesta di ingreso, e informa l'assembler che questa è la fine del codice sorgente. Se l'etichetta "START" è stata inclusa come un'operando della direttiva END, (e voi state usando l'E/A):

linea 42 END START

Allora questo programma comincerà a funzionare appena è stato caricato. Dipendendo dal tipo di applicazione per cui è stato scritto il programma, questo potrà o non potrà essere desiderato.

Alla linea 30, l'indirizzo di ritorno che era stato salvato in @SAVRTN, viene mosso (MOV) a R11. La linea 31 "CLR" (pulisce, cioè, mette a zero tutti i bits) la word in memoria all'indirizzo >837C. >837C è l'indirizzo del byte di STATUS GPL

(Linguaggio di programmazione grafico). In effetti, pulendo questa locazione, voi state dicendo al sistema operativo del computer che ogni cosa è O.K.

La linea 32 DECrementa di due (DECT) R11. "DEC" e "DECT" sono l'opposto di "INC" e "INCT". Essi sottraggono rispettivamente 1 e 2. Diramandosi ad un'indirizzo che è meno di 2 del punto di entrata dell'indirizzo, il computer "congelerà" la risposta visualizzata, finchè non viene premuto FCTN = (QUIT).

Alterando l'indirizzo di ritorno in questa maniera, non è proprio il modo migliore per finire il vostro programma. A questo punto, provate il programma senza l'istruzione DECT R11 della linea 31, e vedrete come è incredibilmente veloce il linguaggio ASSEMBLY del TMS9900. Lo schermo, in un batter d'occhio appena, visualizzerà il programma e terminerà.

La linea 33 usa l'istruzione "RT". Questa istruzione ha lo stesso effetto come B *R11. Esse sono intercambiabili. L'istruzione di ritorno "RT", è più facilmente comprensibile e leggibile. Essa è più mnemmonica (aiuta meglio la memoria) di B *R11. Il programma esegue un salto incondizionato all'indirizzo in R11. Questo completa il programma, e ritorna il controllo del computer al sistema operativo.

Qui ora c'è il listato ASSEMBLY prodotto con l'E/A. Le opzioni usate sono: "R" (etichetta per i registri generali workspace), "L" (che produce il listato), e "C" (che produce il codice oggetto in formato compresso).

==
NOTA DEL CORRETTORE:
Il listato del programma è stampato a parte per una migliore consultazione
Fare riferimento ad esso nel leggere quello che segue
==

La prima colonna di numeri sono i vostri numeri di linea. I valori nella seconda colonna (0000, 0002, 0004, ecc.) rappresentano il contatore di locazione. Con l'E/A, esso inizia da >0000 ed è incrementato all'indirizzo di ogni linea. Il contatore di locazione va da >000A a >000C alla linea 10, perchè la direttiva BSS nella linea 9 mette da parte 2 bytes di memoria che sono parte del programma. Così, >000A + 2 = >000C. Notate che le prime tre direttive non influenzano il valore del contatore di locazione. I valori visti qui sono valori di spiazzamento. Essi sono addizionati all'indirizzo iniziale, dove il codice è caricato, in ordine di indirizzo di ogni linea o etichetta, nel programma.

La terza colonna di numeri sono la rappresentazione in Hex del codice macchina di ogni indirizzo (Valore del contatore di locazione). Alla linea 5, locazione >0002, voi vedrete il valore >000A (decimale 10), che è il valore a cui AMTX viene inizializzato. La linea 16, locazione >0042, mostra il valore >04C4. Questo è l'Hex per l'istruzione in linguaggio macchina, per CLR (resettare, o fissare a zero tutti i bit). La lunghezza del vostro programma può essere determinata sottraendo il valore iniziale del contatore di locazione, dall'ultimo valore nello stesso contatore. In questo caso, >0090 - >0000 =>0090 , indicando che questo programma è lungo >90 bytes (decimale 144).

Qui c'è la tavola dei simboli costruiti dal programma. Ogni simbolo che era usato accanto al suo indirizzo, è mostrato in ordine alfabetico. L'indirizzo dopo ciascun simbolo, può essere l'indirizzo attuale (STATUS a >837C), o il valore del contatore di locazione.

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sulla codificazione.

Dalla sezione 3-3 pag. 46 alla sezione 3-9 pag. 55
Dalla sezione 4-1 pag. 56 alla sezione 5-8 pag. 74

INDIRIZZAMENTO

Ci sono cinque modi di indirizzamento generali. Tre sono usati nel programma esempio.

```
REGISTRO WORKSPACE DIRETTO        MOV  11,6
REGISTRO WORKSPACE INDIRETTO      B    *R11
MEMORIA SIMBOLICA                 A    @AMTX,@AMTY
```

Dalla sezione 14-1 pag. 208, alla sezione 14-1-1 pag..210
Sezione 14-1-4 pag.212
Dalla sezione 14-3 pag. 224, alla sezione 14-4-2 pag. 228
Sezione 14-5-2 pag. 234

DIRETTIVE

Queste sono le direttive usate nel programma esempio:

AORG - Origine assoluta. Influisce sul contatore di locazione.
BSS - Riserva memoria
EQU - Eguaglia. Associa un'etichetta con un'indirizzo.
BYTE - Inizializza 8 bits di memoria per alcuni valori.
DATA - Inizializza 16 bits di memoria per alcuni valori.
TEXT - Inizializza la memoria usando un carattere stringa.
DEF - Definisce. Crea le etichette che definiscono parte del vostro codice oggetto, così che il programma sia utilizzabile per altro software e aggiunge le etichette nella tavola REF/DEF.
REF - Riferimenti esterni. Crea altre etichette di programma utilizzabili per il vostro programma.
END - Fine del codice sorgente.

Dalla sezione 6-1 pag. 78, alla sezione 6-14-2 pag.102

ISTRUZIONI ARITMETICHE

Queste operazioni aritmetiche sono usate, e dettagliate, nel programma esempio:

A - Addiziona Word. Addiziona gli operandi, e mette il risultato nel secondo operando
INC - Incrementa il contenuto dell'operando, di 1.
INCT - Incrementa il contenuto dell'operando, di 2.
DEC - Decrementa il contenuto dell'operando, di 1.

DECT - Decrementa il contenuto dell'operando, di 2.
DIV - Divide il primo operando dal secondo operando.

Ogni istruzione aritmetica influisce il registro STATUS, in base ai risultati dell'operazione

ISTRUZIONI DI SALTO E DIRAMAZIONE
Cinque, delle molte specie di istruzioni di salto e diramazione usate nell'esempio, sono:

B - Diramazione incondizionata, per diramarsi ad un'indirizzo specifico
BL - Diramazione e allacciamento, e mette l'indirizzo di ritorno in R1
BLWP- Diramati e carica il puntatore dello spazio di lavoro (Workspace) si dirama ad una routine, e fissa il registro WP per puntare a quelle routines dello spazio registri.
JMP - Salto incondizionato ad un indirizzo.
JEQ - Salta se uguale. Se il bit di uguaglianza del registro di stato (ST) è 1, allora salta all'indirizzo specificato.

Dalla sezione 8-1 pag. 138, alla sezione 8-3 pag. 143

ISTRUZIONI DI CONFRONTO
Solo un tipo di istruzionei di confronto è usata nel programma esempio. Comunque, esse operano tutte in modo simile.

CI - Confronta immediatamente i contenuti del registro nominato (primo operando), per un valore specifico (secondo operando)

Dalla sezione 10-1 pag. 161, alla sezione 10-5 pag. 168
Sezione 10-9 pag. 172

ISTRUZIONI PER CARICARE E MUOVERE
Esistono 8 istruzioni di questo tipo, ma nel nostro programma ne abbiamo usato solo 4.

LWPI- Carica immediatamente il puntatore workspace, necessario per stabilire registri alternativi di workspace.
MOV - Muove una Word. Copia una Word (16 bits) in un'altra Word.
MOVB- Muove un Byte. Copia un Byte (8 bits) in un'altro Byte.
LI - Carica immediatamente, usato per mettere valori specifici in un registro.

Queste istruzioni influiscono il registro di stato (ST), a seconda del valore all'indirizzo coinvolto.

Sezione 11-5 pag. 184

ISTRUZIONI LOGICHE
Una sola è usata nel programma esempio.

CLR - Pulisce, fissa tutti i bit del registro o indirizzo nominato, a zero.

Dalla sezione 12-1 pag. 194, alla sezione 12-5 pag. 204

ISTRUZIONI DI SPOSTAMENTO DEL REGISTRO WORKSPACE
Nuovamente, solo un'istruzione è utilizzata, ma essa è analoga agli altri tipi di istruzioni di spostamento del registro.

SLA - Spostamento aritmetico a sinistra. Sposta i bits del registro workspace nominato di un specificato numero di posizioni a sinistra, e riempie la parte destra con zeri. Tutte le istruzioni di spostamento influiscono il registro di stato (ST).

Dalla sezione 13-1 pag. 206, alla sezione 13-2 pag. 207

PSEUDO ISTRUZIONI
Anche qui ne viene usata solo una.

RT - Return. Lo stesso risultato di "B *R11"

Sezione 19-2 pag. 307

LEGGERE QUESTI TERMINI SUL GLOSSARIO

Operatori aritmetici
Campo commento
Costante
Context Switch (Commuttatore del contesto)
CPU RAM
Operando destinazione
Direttive
Espressioni
Istruzioni immediate
Indirizzamento di memoria indicizzato
Istruzioni di salto
Campo etichetta
Contatore di locazione
Codice mnemonico
Campo OP-CODE (codice operativo)
Operando
Campo operando
Simboli predefiniti
Pseudo istruzioni
Simboli
Tavola dei simboli
Indirizzi simbolici
Indirizzamento simbolico della memoria
VDP RAM
Indirizzamento dei registri Workspace.

CAPITOLO SESTO

ASSEMBLARE E FAR GIRARE UN PROGRAMMA.
(con l'editor/assembler)

Quello che segue è una breve descrizione dei passi che dovrete fare per inserire, assemblare, salvare, e far girare il programma esempio del capitolo precedente.

Leggere, nella documentazione che accompagna il package dell'E/A, la parte riguardante l'EDITOR, per la preparazione dei programmi sorgente. Occorre un pò di empo per prendere confidenza con le capacità di EDITING che sono disponibili. Molte delle caratteristiche del programma EDITOR si possono applicare molto bene per la reparazione di testi e documenti, quanto per le istruzioni sorgenti ASSEMBLY.

I dati presentati, presumono che voi abbiate un solo disk drive, ma se ne avete iù di uno, allora lasciate il dischetto con il programma dell'E/A nel driver N°.1, e salvate la vostra sorgente sul drive N°.2 o 3.

Ottenete il titolo principale di schermo dell'E/A, e con il dischetto del oftware nel driver 1, selezionate l'opzione "EDIT" (N°.1). Questo vi presenta il enù dell'EDITOR, da cui scegliete l'opzione EDIT (N°.2).

Il programma EDITOR si caricherà dal dischetto. Battete attentamente il rogramma di esempio, usando solo lettere MAIUSCOLE. Controllate quello che battete. l programma è stato assemblato e fatto girare con successo, sia con l'E/A che con la M. Se a voi non funziona è perchè avete fatto degli errori nel batterlo.

Quando avete finito di inserire il codice sorgente, premete "ESCAPE" (FCTN 9) due olte, per ritornare al menù dell'EDITOR. Poi inserite il discheto su cui volete alvare la vostra sorgente, nel disk driver scelto, e selezionate l'opzione "SAVE" N°.3). Il "PROMPT" (Richiamo, suggerimento) VARIABLE 80 FORMAT (Y/N)? apparirà. Se oi battete "Y" il formato sarà VARIABLE, mentre se battete "N", esso sarà FIXED.

L'E/A tratta FILE di lunghezza sia VARIABLE, che FIXED. In questo esempio si e celto la lunghezza VARIABLE, per cui battete "Y". Poi inserite il nome con il quale olete chiamare il programma sorgente. Per questo esempio battete "DSK1.SOURCE", e nfine premete ENTER. I contenuti del BUFFER testo (il programma SOURCE che avete opena inserito) saranno salvati sul drive N°.1, in record di formato variabile lungo 0 byte, come "DSK1.SOURCE".

Allora togliete il dischetto, e inserite quello con il programma Assembler ell'E/A. Premete escape (FCTN 9) per visualizzare il menù dell'E/A, e selezionate 'opzione "ASSEMBLE" (N°.2).

A questo punto vi sarà chiesto se volete caricare l'ASSEMBLER. Ciò è fatto per ermettervi di controllare ed assicurarvi che nel driver vi sia il dischetto giusto. e tutto è O.K. rispondete "Y" al prompt.

Il prossimo prompt che vedrete vi chiederà il nome del file che contiene il ostro programma sorgente. Rimuovete il dischetto che contiene l'assembler, ed nserite quello in cui avete salvato la sorgente. Battete "DSK1.SOURCE". Apparirà un ltro prompt che vi chiederà il nome che vorrete dare al codice oggetto generato dal odice sorgente. In questo caso battete "DSK.OBJECT". Ancora un prompt per ichiedervi un nome valido di un dispositivo per il listato dell'ASSEMBLY.

Se non avete la stampante, voi potete salvare il listato su disco per rivederlo uccessivamente con il programma EDITOR, oppure decidere di non avere un listato ell'ASSEMBLY. Se invece avete una stampante, dovrete inserire i parametri che

descrivono l'interfaccia che state usando (Per Esempio: RS232.BA=1200, RS232/2.BA=9600.PA=N, PIO)

Se decidete di non avere un listato stampato, o su disco, premete ENTER quando appare questo prompt.

L'opzione per produrre o non produrre un listato sarà indicato dal prossimo prompt, il quale vi chiderà le opzioni desiderate per questo ASSEMBLY. L'opzione "R" permette di riferirsi ai registri generali workspace nel programma come R0, R1, R2, R10, R11, ..etc. L'opzione "C" produrrà il codice ogetto in formato compresso, il quale occupa meno memoria del codice oggetto non compresso. L'opzione "L" è quella che vi permette di avere un listato dell'ASSEMBLY. Se voi non scegliete questa opzione, nessun listato sarà prodotto. "S" è l'opzione che includerà una mappa della TAVOLA dei SIMBOLI, se avrete selezionato anche "L".

Per questo esercizio la minima opzione di cui avrete bisogno è "R". Scgliete le altre opzioni in base alla configurazione hardware che possedete .

Esiste un'altra opzione supplementare disponibile, che non è documentata sul manuale E/A, ed è "T". Questa stamperà le locazioni e il valore Hex di ogni byte di una stringa TEXT. Senza l'opzione "T" il listato ASSEMBLY stamperà solo l'indirizzo di inizio (valore del contatore di locazione), di una stringa TEXT, e il valore Hex solo del primo byte di quella stringa.

"ASSEMBLER EXECUTING" (Esecuzione assembler) dovrebbe ora apparire, mentre l'assembler elabora il vostro codice sorgente. Se ogni cosa è stata fatta bene, il processo ASSEMBLY dovrebbe terminare con "0000 ERRORS". Se vi sono degli errori, tornate indietro e riesaminate il codice sorgente che avete salvato come "DSK1.SOURCE". Per editare il codice sorgente, mettete il dischetto dell'E/A nel disk driver N°.1. Ottenete il titolo di schermo EDITOR, e poi selezionate LOAD (N°.1). Dopo che l'editor è stato caricato, vi sarà il prompt per il nome del file. A questo punto voi togliete il dischetto dell'E/A dal drive, e inserite quello contenente il vostro programma sorgente, e poi battete "DSK1.SOURCE". L'editor caricherà il programma. Selezionate infine l'opzione EDIT (N°.2) per rivedere ed editare il vostro programma sorgente. Quando avrete finito di correggere, risalvate il codice sorgente sotto lo stesso nome, e ripetete il processo ASSEMBLY.

Una volta che avrete assemblato con successo il programma campione, premete ENTER per ritornare allo schermo dell'E/A, e selezionate l'opzione "LOAD and RUN" (N°.3). Il primo prompt vi chiederà il nome del file che contiene il vostro programma oggetto. Voi battete "DSK1.OBJECT", ed il LOADER (Caricatore) lo caricherà dal dischetto nella memoria. Apparirà ora un'altro prompt di richiesta per il "FILE NAME" (Nome del file). Questo è perchè il loader dell'E/A vi permette di continuare a caricare programmi oggetto fino che la memoria è piena. Siccome noi abbiamo un solo programma da caricare, a questo punto, premete ENTER. Il prossimo prompt sarà per il "PROGRAM NAME" (Nome del programma). Il nome, ed il suo indirizzo del punto di entrata era DEFinito come "START". Battete START , e finalmente (speriamo...) il programma dovrebbe girare. Se voi codificate l'ultima linea del programma come "END START", allora esso sarà eseguito immediatamente appena caricato, senza che voi dobbiate dare l'indirizzo del punto di entrata.

I numeri "43" appariranno nell'angolo inferiore sinistro dello schermo. Per uscire dal controllo del programma, premete "QUIT" (FCTN =). Ritornerete così al titolo principale dello schermo.

Ricordate che è possibile aver fatto un errore inserendo il programma esempio, e terminare tuttavia con "0000 ERRORS" alla fine del processo ASSEMBLY. Quando il programma ASSEMBLY gira, esso è sotto il controllo del computer, e può essere necessario spegnerlo e riaccenderlo, per riprendere il controllo di un programma ribelle.

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni su come assemblare e far girare i programmi.

Dalla sezione 1-1 pag.15, alla sezione 2-5 pag.38
Dalla sezione 15-1 pag.235, alla sezione 15-1-1 pag.236
Dalla sezione 15-5 pag.243, alla sezione 15-5-2 pag.245
Dalla sezione 19-1 pag.305, alla sezione 19-2 pag.307

GUARDA QUESTI TERMINI SUL GLOSSARIO

Assembler
Opzioni Assembler
Modo comando
Modo Edit
Marcatore "Fine del file"
Errore fatale (Errore grave)
Lista del file
Loader (Caricatore)
Loading
Errore non fatale
Tasti speciali
Simboli
Tavola dei simboli
Window (finestra)

CAPITOLO SETTIMO

SCHERMO E VISUALIZZAZIONE DEI CARATTERI

Una delle cose che potete fare nella maniera giusta, con un linguaggio come il TI BASIC, è di visualizzare numeri,lettere, ed altri simboli sullo schermo, e controllare la loro posizione e colore. Il programma esempio introduce due routines fondamentali della VDP. I numeri "43", erano visualizzati nell'angolo inferiore sinistro dello schermo, ed essi apparivano come simboli neri su uno sfondo verde. Questi sono i colori di DEFAULT dello schermo, ed il set di caratteri che sono eseguiti quando l'ASSEMBLY gira. Questo capitolo esaminerà più da vicino il linguaggio ASSEMBLY del TMS9900 circa la visualizzazione sullo schermo.

Il processo necessario per generare i segnali video che creano i simboli, e i grafici, è trattato da un Microprocessore separato, il TMS9918A.

Il processore centrale, come sapete è il TMS9900, ed è quello che tratta direttamente con il linguaggio ASSEMBLY. Attualmente il nostro TI-99/4A, contiene parecchi microprocessori diversi, che lavorano di comune accordo, per permettere al computer di fare tutto quanto voi sapete....che è tanto.

L'area della memoria indicata per il Video Display Processor (VDP), è necessita di una speciale area separata, conosciuta come VDP Random Access Memory (RAM).

Le istruzioni ASSEMBLY come MOVE, o ADD non lavoreranno su indirizzi dentro la VDP RAM. I dati devono essere manipolati dentro il dominio della CPU RAM, e scritti o letti dalla VDP RAM per mezzo di routine speciali, che permettono a questi due processori di spartire i dati, e comunicare l'un l'altro. Il primo programma esempio ha introdotto due routines VDP molto utili, e cioè: VSBW (VDP Single Byte Write), e VMBW (VDP Multiple Byte Write), che ricopiano i dati del vostro programma nella VDP RAM. I dati erano prima fissati dentro al programma, poi mettendo certi valori richiesti nei propri registri, e diramati agli indirizzi delle routines, lo schermo era pulito, e poi la somma di 10 più 33 era visualizzata.

Questo modello di caricamento dei registri chiave con certi valori di parametri, diramandosi a indirizzi speciali, ricorre durante tutte le varie routines VDP

Per usare qualsiasi routine VDP nel vostro programma dovete includere una direttiva REF, con il nome simbolico di ogni routine che volete usare.

Uno dei modi più facili per visualizzare titoli e messaggi, combina l'uso della direttiva TEXT e VMBW. Con l'E/A questo indirizzo simbolico della routine è eguagliato appunto a VMBW. Con la direttiva TEXT la memoria può essere inizializzata facilmente con stringhe di caratteri leggibili. Il modo in cui la stringa appare nel programma è esattamente come essa apparirà quando sarà correttamente visualizzata. Ecco qui una parte di programma che esegue questo:

```
MSG1 TEXT '** PROGRAMM NUMBER 2 **'
```

La stringa dei caratteri è racchiusa dai segni di virgolette semplici ('), ed esse non faranno parte di ciò che sarà visualizzato. Qualsiasi carattere può essere nella direttiva TEXT, escluso il simbolo della virgoletta singola. Essa può essere solo usata per delimitare i contenuti della stringa. Per visualizzare questo titolo del programma, voi dovete dare alla VMBW tre tipi di informazioni:

1°) L'indirizzo in VDP RAM in cui vorreste che questo messaggio venga messo cioè scritto (La posizione sullo schermo)

2°) L'indirizzo iniziale del messaggio. Dove trovare il messaggio.

3°) Il numero dei byte da scrivere. La lunghezza del messaggio.

In TI BASIC lo schermo può essere indirizzato come riga N, e colonna N. Ci sono 24 righe e 32 colonne. Questa stessa configurazione dello schermo è definita nel linguaggio ASSEMBLY come una tavola di 768 Bytes. Questa è la TAVOLA IMMAGINE dello SCHERMO (TIS) nella VDP RAM.

Ogni Byte di questa tavola rappresenta una posizione dello schermo. Gli indirizzi nella VDP RAM vanno da 0 a 767 decimale, e da >0000 a >02FF Hex. Usate questa formula per determinare il giusto indirizzo in VDP RAM per un dato gruppo di valori di righe o colonne.

INDIRIZZO DECIMALE VDP RAM = ((RIGA - 1) * 32) + (COLONNA - 1)

Per visualizzare il messaggio di cui sopra alla riga 10 e colonna 6 :

INDIRIZZO = ((10 - 1) * 32) + (6 - 1)
= (9 * 32) + 5
= 288 + 5
= 293

Per passare questa informazione al VMBW, metterla nel registro 0 con un'istruzione "LI" (carica immediatamente)

```
LI R0,293
```

Il prossimo tipo di informazione richiesto dal VMBW è l'indirizzo iniziale del messaggio. L'etichetta "MSG1" era inclusa con la direttiva TEXT, e viene perciò eguagliata al valore dell'indirizzo iniziale del messaggio. Ricordate che questo valore è L'INDIRIZZO del messaggio, e NON il dato stesso. La routine VMBW necessita di questa informazione nel registro 1. Perciò, usando ancora l'istruzione "LI", avremo:

```
LI R1,MSG1
```

Ed alla fine,VMBW necessita di sapere quanto è lungo il messaggio in bytes. Occorre un byte per ogni carattere, del messaggio (spazi compresi). Siccome ci sono 22 caratteri in MSG1, questo valore deve essere messo nel registro 2:

```
LI R2,22
```

Poichè il contatore di locazione (PC) avanza sempre ad un'indirizzo di word pari, è sempre una buona idea mettere da parte, memorizzando con una direttiva come TEXT, un totale pari di bytes.

Per visualizzare il messaggio, eseguire una BLWP all'indirizzo della routine VMBW:

```
BLWP @VMBW
```

```
     REF  VMBW
MSG1 TEXT '** PROGRAM NUMBER 2 **'
DISP LI   R0,293
     LI   R1,MSG1
     LI   R2,22
     BLWP @VMBW
```

Naturalmente la direttiva TEXT è soltanto uno dei modi per costruire dati visualizzabili. Voi potete anche usare i codici ASCII per lettere e numeri, e creare un programma con stringhe alfanumeriche e numeri, proprio come il programma esempio fece con il risultato del suo problema di addizione.

```
PNTANS  BSS 2
        MOV  R5,@PNTANS
        MOVB R5,@PNTANS
PUTUP   LI   R0,768
        LI   R1,PNTANS
        LI   R2,2
        BLWP @VMBW
```

La routine VSBW (VDP single byte write), scrive un solo byte alla volta. Poichè la lunghezza del dato da scrivere è sempre 1, il registro R2 non è necessario per la VSBW. Voi dovrete dare solo il corretto indirizzo alla VSBW, in R0, e il dato da scrivere nei primi 8 bits (bytes sinistro della word) del registro 1. Questo è diverso dalla routine VMBW in cui il registro 1 conteneva l'indirizzo del dato.

Supponete che al posto del titolo all'indirizzo MSG1, voi vogliate visualizzare il simbolo dell'asterisco (*), alla riga 10 e colonna 6. Il suo codice ASCII è 42 in decimale, >2A in Hex. Voi avete già calcolato il suo indirizzo in VDP RAM come 293, perciò:

```
        LI   R0,738
        LI   R1,>2A00
        BLWP @VSBW
```

Una istruzione BASIC che dovrebbe esservi familiare è CALL SCREEN(n). Il valore di "n" è un numero tra 1 e 16, ed ogni numero rappresenta un colore diverso dello schermo.

Nel linguaggio ASSEMBLY del TMS9900, è disponibile lo stesso set di colori, ma i loro valori vanno da 0 a 15 in decimale, >0 a >F Hex. Il colore del bordo dello schermo è controllato dal registro 7 di sola scrittura della VDP RAM. L'accesso a questo registro, e il controllo del colore del bordo dello schermo è compiuto per mezzo della subroutine VDP Write To Register (Scrivi nel registro VDP), eguagliata dall'E/A a "VWTR". I colori, ed il loro valore in codice Hex, sono:

TRASPARENTE	>0	ROSSO	>8
NERO	>1	ROSSO CHIARO	>9
VERDE	>2	GIALLO SCURO	>A
VERDE CHIARO	>3	GIALLO CHIARO	>B

BLU SCURO	>4	VERDE SCURO	>C
BLU CHIARO	>5	MAGENTA	>D
ROSSO SCURO	>6	GRIGIO	>E
CIANO	>7	BIANCO	>F

Per fissare il colore del bordo dello schermo al MAGENTA, l'istruzione allora sarà:

```
LI      R0,>070D
BLWP    @VWTR
```

Il registro 0 contiene tutte le informazioni che servono al VWTR. Il byte sinistro (leggendo da sinistra a destra, la prima e la seconda cifra Hex, >07), del registro 0, dice al VWTR in quale registro scrivere. In questo esempio, nel registro VDP 7.

I registri VDP sono registri di UN SOLO BYTE (8 bit), a differenza dei registri generali workspace, che sono registri di Word, cioè 2 Bytes, o 16 Bits. Il Byte destro, (cifra Hex >0D) contiene il valore del colore che volete, (in questo caso il magenta). Di questi 8 Bits, i 4 bits meno importanti (cifra Hex >D) fissano il colore del bordo dello schermo. I bits più importanti (cifra Hex >0) fissano il colore di FOREGROUND (Primo piano), quando il PROCESSORE VIDEO TMS 9918A è in MODO TESTO.

Il MODO TESTO, è un'altra forma di visualizzazione disponibile con il processore video TMS9918A. Quando voi raggiungerete un buon livello di competenza con il linguaggio ASSEMBLY del TMS9900, e con l'uso del VDP, voi potrete provare altri modi. Per ora il valore che metterete in questa posizione non ha nessuna importanza. Il modo di visualizzazione in cui il vostro computer opera mentre è in BASIC o EX.BASIC per la maggior parte di applicazioni, è il GRAPHICS MODE (Modo grafico). Approfondire questo modo di visualizzazione prima di provare ad usarne altri.

L'effetto di fissare il registro 7 della VDP RAM a >0D, è di creare una striscia di color magenta nella parte alta e bassa dello schermo. L'istruzione equivalente in BASIC, cioè CALL SCREEN (n), influisce non solo sul colore del bordo, ma su tutto il colore di BACKGROUND (colore di sfondo) di tutto lo schermo. In questa maniera, non importa cos'è visualizzato sullo schermo, ci sarà un solo colore di sfondo, uniforme, per tutti i caratteri.

I set di colori dei caratteri di foreground e background sul vostro computer, sono controllati da un'area della VDP RAM conosciuta come TAVOLA DEI COLORI.

Ogni entrata nella TAVOLA DEI COLORI è composta da un dato di un byte. Ogni byte controlla il colore di foreground e background di un gruppo di 8 caratteri. La Tavola dei Colori è rilocabile, cioè, con certe istruzioni del TMS9900 è possibile cambiare la locazione dentro il VDP RAM che la tavola occuperà. Cambiare le locazioni della Tavola dei colori, è necessario solo per altri modi di visualizzazione. Per adesso non rilocatela, ma usate il suo default, in VDP RAM, che inizia all'indirizzo >0380.

Per cambiare i colori di primo piano e di sfondo di un carattere particolare, deve essere determinato l'indirizzo corrispondente dentro la tavola dei colori, e un byte del dato deve essere messo in quell'indirizzo. Per simulare CALL SCREEN (n), è necessario cambiare il colore di sfondo, mentre si lascia il colore di primo piano al valore di default di >1 (nero). Il valore per nero su magenta dovrebbe essere >1D. I quattro bits di sinistra del byte controllano il colore di foreground (>1 = nero) mentre i quattro bits di destra controllano il colore di background (>D = magenta).

Naturalmente questo dato deve essere messo dentro l'indirizzo corretto della tavola dei colori, per produrre il risultato desiderato. Più sotto vi è una utile tabella che mostra in dettaglio la tavola dei colori. Gli indirizzi dati sono il valore di spiazzamento. Ogni valore della tabella deve essere aggiunto all'indirizzo iniziale della tavola dei colori in VDP RAM. Nel caso di questo esempio, operando in modo grafico, e non avendo fatto niente per rilocare la tavola dei colori, l'indirizzo iniziale in VDP RAM di quest'ultima è >0380.

TABELLA DI RIFERIMENTO DELLA TAVOLA DEI COLORI

==

SPIAZZAMENTO DELLA TAVOLA DEI COLORI	CODICI DEI CARATTERI INFLUITI		
>00	>00	THROUGH	>07
>01	>08		>0F
>02	>10		>17
>03	>18		>1F
>04	>20		>27
>05	>28		>2F
>06	>30		>37
>07	>38		>3F
>08	>40		>47
>09	>46		>4F
>0A	>50		>57
>0B	>58		>5F
>0C	>60		>67
>0D	>68		>6F
>0E	>70		>77
>0F	>78		>7F
>10	>80		>87
>11	>88		>8F
>12	>90		>97
>13	>98		>9F
>14	>A0		>A7
>15	>A8		>AF
>16	>B0		>B7
>17	>B8		>BF
>18	>C0		>C7
>19	>C8		>CF
>1A	>D0		>D7
>1B	>D8		>DF
>1C	>E0		>E7
>1D	>E8		>EF
>1E	>F0		>F7
>1F	>F8		>FF

I codici dei caratteri per l'intera gamma dei caratteri possibili inizia a >00, e finisce a >FF. Riferendosi alla tabella appena riportata, la tavola dei colori indirizza il valore dello spiazzamento per il carattere >FF a >1F. Aggiungendo ognuno dei valori di spiazzamento all'indirizzo iniziale, che come già detto è >0380, si avrà:

```
>0380+          >0380+
>0000=          >001F=
------          ------
>0380           >039F
```

Questo dimostra che per influire sui colori di questa gamma di caratteri, un byte di dati del colore deve essere messo nel VDP RAM negli indirizzi da >0380 a >039F. Poichè un solo byte di dati è richiesto per ogni indirizzo, verrà usata la routine VSBW. Le istruzioni per fare questo sono:

```
       LI    R0,>0380 Carica R0 con il primo indirizzo in VDP RAM
       LI    R1,>1D00 Mette il codice del colore nel byte sinistro di R1
PUTCOL BLWP  @VSBW   Scrive il byte sinistro di R1 all'indirizzo in R0
       INC   R0       Aggiunge 1 all'indirizzo in R0
       CI    R0,>039F Confronta il valore in R0 con >039F
       JLE   PUTCOL   ...Se più basso o eguale, ripete il procedimento
```

L'ultima istruzione usata era "JLE" JUMP if LOW or EQUAL (Salta se più basso o eguale). La precedente, "CI" COMPARE IMMEDIATE (Confronta immediatamente), controlla R0 per l'ultimo valore dell'indirizzo che deve essere scritto. L'istruzione JLE completa il confronto, dirigendo il programma logico al ciclo PUTCOL, finchè il valore dell'indirizzo in R0 è minore o uguale a >039F.

I passi delineati sin qui per influire le entrate nella tavola dei colori, sono indicati per imitare il comando basic CALL SCREEN (n). Un altro comando basic è CALL COLOR (s,f,b), dove "s" rappresenta il set di caratteri da visualizzare, "f" il colore di primo piano, e "b" il colore di sfondo. Il set può essere un numero da 1 a 14. In BASIC questi set di colori sono equivalenti allo spiazzamento della tavola dei colori, valori da >04 a >11. Per specificare una combinazione di colori come "bianco su blu scuro", per l'asterisco (cod. car. = 42 dec. o >2A hex.), dovrebbe essere usata l'istruzione BASIC: CALL COLOR(2,16,5). L'asterisco fa parte del set di caratteri N°.2, in TI BASIC, i cui codici dei colori per il bianco e blu scuro sono rispettivamente 16 e 5.

Per realizzare lo stesso risultato nel linguaggio ASSEMBLY del TMS9900, per prima cosa consultate la tabella di riferimento dei colori di più sopra, e poi cercate la gamma dei valori del codice dei caratteri a cui appartiene l'asterisco. Siccome il suo codice Hex ">2A" cade dentro la gamma dei valori da >28 a >2F, lo spiazzamento del valore sarà di >05. allora aggiungete questo valore di spiazzamento al valore iniziale della tavola dei colori, per determinare l'indirizzo corretto. (>0380 + >05 = >0385).

I valori dei colori, nell'ASSEMBLY del TMS9900 sono tutti UNO in meno del loro valore in BASIC. Il bianco è 15 dec. hex >F, il blu scuro è 4 dec. o >4 hex. Le istruzioni per fissare il colore dell'asterisco in bianco su sfondo blu scuro, allora

saranno:

```
     LI     R0,>0385 Indirizzo della tavola dei colori, per l'asterisco
     LI     R1,>F400 Valore dei colori, Bianco "F", e blu scuro "4"
     BLWP   @VSBW    Scrive un byte di dati
```

Nessun ciclo di programma è coinvolto, perchè quest'esempio influisce solo sui colori di primo piano e sfondo di un set di caratteri. Notare dalla tavola di riferimento dei colori, che vi sono entrate per codici inferiori e maggiori della gamma dei caratteri ASCII (da 30 a 126 dec. da >1E a >7E hex). Questi valori non sono definiti come caratteri visualizzabili. Alcuni rappresentano codici ASCII di controllo, che regolano le comunicazioni tra computer. Per mezzo del linguaggio ASSEMBLY del TMS9900, voi potete far uso di tutti questi valori dei caratteri in varie maniere.

Per prima cosa richiamate il programma esempio, e la routine CLEAR. Il metodo standard per pulire lo schermo, è di riempirlo interamente di spazi (cod. dec.32, >20 hex). Ma la routine CLEAR del programma d'esempio scriveva il codice del carattere >00 sullo schermo...e questo è perchè il suo codice non è definito come simbolo visualizzabile, per cui il suo effetto è lo stesso come se avessimo usato il codice del carattere spazio.

Un altro modo di usare questi codici extra dei caratteri è per i grafici a colori. Ricordate, dal TI BASIC, che se i colori di foreground e background di un carattere sono fissati allo stesso colore, ogniqualvolta che il carattere è visualizzato, apparirà un unico blocco di colore. Un sistema per applicare questo, potrebbe essere la creazione di un bordo colorato attorno allo schermo. Ogni riga dello schermo contiene 32 colonne. Per creare questo bordo usate le colonne 1,2 e 31,32 di ogni riga. Fissare i caratteri usati per riempire queste colonne allo stesso colore del bordo della parte alta e bassa dello schermo. Per fare in maniera da poter ancora visualizzare e usare il set di caratteri standard ASCII, usate dei caratteri che che sono fuori dalla gamma di quelli visualizzabili, o che sono poco usati.

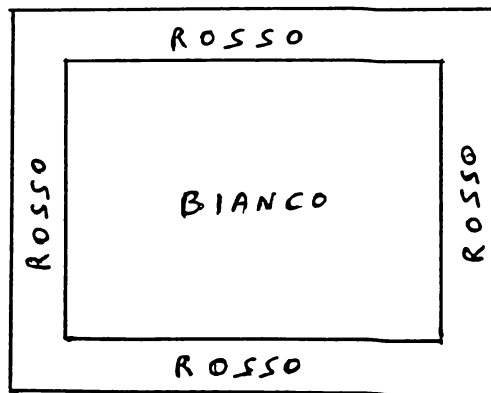
Qui sotto vi sono le istruzioni per l'ASSEMBLY del TMS9900. Notate che la direttiva "DATA" può essere usata per inizializzare più di una parola alla volta.

```
     BORDER  DATA >8080,>2020,>2020,>2020
             DATA >2020,>2020,>2020,>2020
             DATA >2020,>2020,>2020,>2020
             DATA >2020,>2020,>2020,>8080

     *Definisce 32 caratteri da usare per riempire
     *ogni riga dello schermo. I caratteri che occupano
     *le colonne 1,2 e 31,32 sono maggiori di qualsiasi
     *codice ASCII, e caratteri nelle rimanenti colonne sono
     *SPAZI (dec.32 >20hex).

             LI   R0,>0706 Fissa la parte alta e bassa al blu scuro.
             BLWP @VWTR
             LI   R0,>0390 Fissa il colore del carattere >80 al rosso
                           scuro su rosso scuro.
             LI   R1,>6600
```

```
        BLWP @VSBW
        LI   R0,>0383 Fissa la gamma dei caratteri ASCII come nero su
                      bianco. Il carattere spazio (>20) apparirà
                      bianco.
        LI   R1,>1F00
CLOOP   BLWP @VSBW
        CI   R0,>038F
        JEQ  BPUT
        INC  R0
        JMP  CLOOP
BPUT    LI   R0,0      Riempie lo schermo con il pattern del bordo.
        LI R1,BORDER
        LI   R2,32
BLOOP   BLWP @VMBW
        CI   R0,736
        JEQ  EXIT
        AI   R0,32     Addiziona immediatamente 32 a R0, per
                       indirizzare la nuova riga.
EXIT    ...........

(resto del programma)
```

I pattern usati per generare i caratteri, sono controllati da un'altra tavola della VDP RAM, chiamata TAVOLA DESCRITTRICE DEI PATTERN, (TDP) ed è anch'essa rilocabile. Il default dell'indirizzo iniziale in VDP RAM della Tavola descrittrice dei pattern, è >0800. Ogni entrata della tavola occupa 8 bytes. Cambiando i valori memorizzati nella suddetta tavola, potrete così crearvi i vostri grafici o simboli personali. Potrete ridefinire il set di caratteri ASCII, o usare qualsiasi altro carattere disponibile. Per una completa spiegazione su come creare i pattern, riferitevi ai manuali TI BASIC, o EX/BASIC, sottoprogrammi CALL CHAR.

L'accesso alla Tavola Descrittrice dei Pattern, è molto simile a quello della Tavola dei Colori. Per ridefinire un carattere, o creare un grafico, i dati che descrivono il pattern devono essere messi nel corrispondente indirizzo nella "TDP", con il codice del carattere usato. Per aiutarvi a partire, vi è qui una lista parziale di spiazzamento dei valori della "TDP", ed i loro rispettivi codici dei caratteri. Con un piccolo calcolo voi sarete in grado di determinare l'indirizzo che volete.

TABELLA DI RIFERIMENTO DELLA TAVOLA DESCRITTRICE DEI PATTERN

==

DESCRITTORE PATTERN SPIAZZAMENTO TAVOLA	CARATTERE CODICE INFLUITO
>000	>00
>008	>01
>0F0	>1E
>100	>20
>108	>21
>110	>22
>150	>2A
>158	>2B
>160	>2C
>168	>2D

Se voi moltiplicate il valore del codice del carattere per 8, voi troverete il valore di spiazzamento nella "TDP" per quel carattere. Questo valore deve essere aggiunto all'indirizzo iniziale in VDP RAM per la "TDP". A meno che voi rilochiate la "tavola" per mezzo di istruzioni speciali, essa inizierà a >0800 Hex, 2048 dec. Per creare un nuovo simbolo del cursore, per esempio, voi dovrete per prima cosa moltiplicare il codice del carattere del cursore che volete ridefinire, per 8:

	HEX	DECIMALE
CURSORE =	>1E *	30 *

```
>8 =          8 =
----        -----
>F0          240
```

Poi aggiungere questo risultato all'indirizzo iniziale in VDP RAM.

```
Indirizzo iniziale PDT    >0800 +         2048 +
Valore di spiazzamento    >00F0 =          240 =
                          ------          -----
Indirizzo desiderato      >08F0           2288
```

Adesso voi sapete che l'indirizzo di entrata della TDP, per il pattern del nuovo cursore è >08F0 Hex, Dec. 2288. Le istruzioni dell'ASSEMBLY TMS9900 per cambiare il pattern del cursore, sono:

```
CURPAT  DATA >007E,>4242,>4242,>7E00

        *Definisce 8 nuovi bytes di dati
        *per descrivere il nuovo pattern.

        LI   R0,>08F0  Carica l'indirizzo VDP RAM in R0.
        LI   R1,CURPAT Carica l'indirizzo dei dati in R1
        LI   R2,8      Carica la lunghezza dei dati in R2
        BLWP @VMBW
```

Adesso, il simbolo del cursore è visualizzato con un set di istruzioni simile a questo:

```
        LI   R0,293
        LI   R1,>1E00
        BLWP @VSBW
```

Il pattern che è stato definito, sarà visualizzato al posto del simbolo standard del cursore. Notate che il codice del carattere per il cursore >1E, è ancora usato. Il computer prende il codice del carattere da voi specificato, guarda nella tavola dei colori per, il corretto colore di primo piano e sfondo, poi guarda nella PDT, per il pattern da visualizzare, ed infine visualizza la giusta combinazione pattern/colore all'indirizzo specificato, sullo schermo.

Visualizzare lo schermo ed i caratteri, creati per mezzo del linguaggio ASSEMBLY del TMS9900, non è difficile, una volta che siete padroni degli schemi fondamentali di questo capitolo. La sorprendente velocità del linguaggio ASSEMBLY diventa evidente quando è usata con le applicazioni del VDP.

I cambi dello schermo visualizzato avvengono quasi istantaneamente. Maggiori capacità grafiche diventano accessibili per mezzo del TMS9900, che sono anche possibili in BASIC.

Ecco qui sotto un completo programma in linguaggio ASSEMBLY del TMS9900, che vi dimostrerà alcuni dei principi inclusi in questo capitolo.

```
          DEF   START
          REF   VWTR,VSBW,VMBW
WR        BSS   >20
```

```
RETURN   BSS   2
STATUS   EQU   >837C
BORDER   DATA  >8080,>2020,>2020,>2020
         DATA  >2020,>2020,>2020,>2020
         DATA  >2020,>2020,>2020,>2020
         DATA  >2020,>2020,>2020,>8080
MSG1     TEXT  '** PROGRAM NUMBER 2 **'
START    MOV   R11,RETURN Salva l'indirizzo di ritorno
         LWPI  WR         Carica il puntatore Workspace
         LI    R0,>0706   Fissa il colore del bordo come rosso scuro
         BLWP  @VWTR
         LI    R0,>0390   Fissa il colore del carattere >80 come
                          rosso scuro su rosso scuro
         LI    R1,>6600
         BLWP  @VSBW
         LI    R0,>383    Fissa la gamma dei colori dei caratteri
                          ASCII da visualizzare, come nero su bianco.
         LI    R1,>1F00
CLOOP    BLWP  @VSBW
         CI    R0,>038F
         JEQ   BPUT
         INC   R0
         JMP   CLOOP
BPUT     LI    R0,0       Carica la TIS con il pattern del bordo
         LI    R1,BORDER
         LI    R2,32
BLOOP    BLWP  @VMBW
         CI    R0,736
         JEQ   EXIT
         AI    R0,32
         JMP   BLOOP
EXIT     LI    L0,293     Visualizza il titolo del programma
         LI    R1,MSG1
         LI    R2,22
         BLWP  @VMBW
         CLR   @STATUS   Pulisce il byte del GPL status
         MOV   @RETURN,R11 Indirizzo di ritorno
         DECT  R11        Altera l'indirizzo di ritorno
         RT               Ritorno
         END
```

Seguire le istruzioni del capitolo 6 per assemblare e far girare questo programma. Poichè questo programma altera l'indirizzo di ritorno, come è stato fatto dal primo programma di esempio, esso si congelerà, per permettervi di osservare il risultato. Per uscire dal programma premere FCTN= (QUIT).

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sulla visualizzazione dello schermo e dei caratteri.

Dalla sezione 16-1 pag. 246 alla sezione 16-1 pag. 248
Dalla sezione 21-1 pag. 325 alla sezione 21-2-3 pag. 330
Dalla sezione 21-7 pag. 342 alla sezione 21-7-1 pag. 342
sezione 24-7 pag. 428

GUARDA QUESTI TERMINI SUL GLOSSARIO

Costanti caratteri
Set caratteri
Stringhe caratteri
Tavola dei colori
Caratteri non visualizzabili
Utilità
VDP RAM

CAPITOLO OTTAVO

ELABORAZIONE DELL'INGRESSO DA TASTIERA

L'accettazione e l'elaborazione dei dati inseriti dall'utente, attraverso la tastiera, è sempre una funzione importante del programma. L'accettazione dell'ingresso da tastiera implica anche la migliore visualizzazione dello schermo. Tutte le volte che voi inserite un dato premendo i tasti, vi aspettate di vedere i caratteri che voi avete battuto, visualizzati sullo schermo così come sono stati inseriti. In più, inserendo i dati, vi state abituando ad usare le speciali combinazioni dei tasti funzione per controllare le operazioni del computer.

Molti metodi per realizzare questo sono disponibili in TI BASIC. INPUT X è un'istruzione che può essere usata per inserire un valore numerico. CALL KEY(x,y,z) è un'altra istruzione che può rivelare specifici tasti premuti. Questi semplici comandi in TI BASIC, sono capaci di eseguire incarichi che sono molto più complessi di come appare dalla sintassi richiesta dal TI BASIC

Il linguaggio ASSEMBLY del TMS9900 che vi avvicina a questi obbiettivi implica la lettura e scrittura in VDP RAM, e l'uso di una speciale routine chiamata KEYBOARD SCAN UTILITY (Utilità Scansione della Tastiera). Questa routine è accessibile includendo una direttiva REF KSCAN quando si programma con l'E/A, o EQUagliando questa routine all'indirizzo >6020. Come la maggior parte delle routine precedentemente discusse, KSCAN deve usare l'istruzione BLWP per attivare l'utilità.

Oltre alla routine stessa, vi sono anche alcuni indirizzi speciali che avrete bisogno di conoscere sopra l'uso effettivo di KSCAN. Il valore del byte all'indirizzo >8374 controlla quale dispositivo della tastiera deve essere esplorato. Un valore di >00 esplora l'intera tastiera. Un valore di >01 esplora la parte sinistra della tastiera, incluso il Joystik N°.1. I valori del Joystik sono messi agli indirizzi >8376 (valore Y), e >8377 (valore X). Un valore di >02 all'indirizzo >8374 esplora la parte destra della tastiera, ed il Joystik N°.2. I valori del Joystik N°.2 sono messi agli stessi indirizzi del N°.1 (>8376,>8377). Il valore normale o di default a >8374 è >00, (controlla cioè l'intera tastiera). Un altro indirizzo è>837C, già usato in precedenza, ed è l'indirizzo dello STATUS byte GPL. Ogni volta che un tasto è premuto, e che è diverso dal tasto premuto l'ultima volta che KSCAN è stata chiamata, il BIT 2 dello STATUS byte GPL viene fissato a 1. Il valore del tasto premuto è messo all'indirizzo >8375. Se nessun tasto era stato premuto, i'ndirizzo >8375 contiene >FF.

In attesa che i dati vengano inseriti nel computer, voi vedrete il cursore sullo schermo che segna l'inizio del campo di input. Come il dato è inserito, il cursore si muove alla destra, e il dato stesso viene visualizzato dov'era il cursore. Per ultimo, se viene fatto uno sbaglio inserendo il dato, voi dovrete essere in grado di tornare indietro, e reinserire il dato stesso, finchè ENTER non è premuto.

C'è qui una subroutine fondamentale per eseguire quanto sopra detto. Il suo nome è "CURSOR". Esso presume che la direttiva EQUate sia stata inclusa per KEYADR EQU >8374, KEYVAL EQU >8375, STATUS EQU >837C, e che le direttive BYTE siano incluse per ENTER BYTE >0D, LEFTV BYTE >08, RITEV BYTE >09, ANYKEY BYTE >20.

SUBROUTINE "CURSOR"

```
INPUT:   R0  = Indirizzo di risposta dello schermo.
         R10 = Massima lunghezza della risposta.

OUTPUT:  R9  = Ultimo valore del tasto premuto.
         R7  = Lunghezza attuale della risposta.
             (i dati della risposta iniziano a R0
             per una lunghezza di R7).

01 CURSOR   CLR   R9
02          MOV   R10,R10
03          JEQ   SCAN
04          CLR   @KEYADR
05          LI    R1,>1E00
06          BLWP  @VSBW
07          MOV   R0,R8
08          A     R8,R10
09          MOV   R8,R7
10 SCAN     CLR   @STATUS
11          BLWP  @KSCAN
12          CB    @ANYKEY,@STATUS
13          JNE   SCAN
14          MOV   R10,R10
15          JNE   ENTCHK
16          RT
17 ENTCHK   CB    @ENTERV,@KEYVAL
18          JEQ   ENTER
19          CB    @LEFTV,@KEYVAL
20          JEQ   LEFT
21          CB    @RITEV,@KEYVAL
22          JEQ   RITE
23          C     R7,R10
24          JEQ   SCAN
25          MOV   R7,R0
26          MOVB  @KEYVAL,R1
27          MOVB  @KEYVAL,R9
28          BLWP  @VSBW
29          INC   R7
30 CURPUT   MOV   R7,R0
31          LI    R1,>1E00
32          BLWP  @VSBW
33          B     @SCAN
34 LEFT     C     R7,R8
35          JEQ   SCAN
36          MOV   R7,R0
37          LI    R1,>2000
38          BLWP  @VSBW
39          DEC   R7
40          JMP   CURPUT
41 RITE     C     R7,R01
42          JEQ   SCAN
```

```
43            MOV   R7,R0
44            LI    R1,>2000
45            BLWP  @VSBW
46            INC   R7
47            JMP   CURPUT
48 ENTER      LI    R1,>2000
49            MOV   R7,R0
50            BLWP  @VSBW
51            S     R8,R7
52            RT
```

Prima di analizzare questa subroutine, c'è quì spiegato come voi dovreste usarla nel vostro programma. Prima di eseguire una BL CURSOR, mettete nel R0 l'indirizzo iniziale dello schermo dove voi volete far apparire l'INPUT da tastiera. Nel R10 mettete la massima lunghezza del dato da essere accettato. Se voi volete simulare un "PRESS ANY KEY" = (premi qualsiasi tasto), mettete un valore di lunghezza 0 nel R10. La routine CURSOR ritornerà al vostro programma appena un tasto qualsiasi viene premuto, senza accettare nessun dato, appunto.

Molto spesso, quando voi volete che l'utente del computer risponda ad un PROMPT = (suggerimento), la risposta data ha il valore di una cifra. Un esempio di questo è quando l'utente deve rispondere "Y" o "N", o scegliere un numero o una lettera da un "MENU". CURSOR mette sempre il valore dell'ultimo tasto premuto prima di premere ENTER, nel byte sinistro di R9. Per la risposta di una cifra, il valore della risposta è disponibile in R9 dopo il ritorno dal CURSOR, senza qualsiasi movimento o altre manipolazioni. R7 conterrà la lunghezza attuale del dato che era stato inserito. L'attuale numero dei caratteri battuti possono o non possono essere gli stessi del massimo permesso.

```
01 CURSOR   CLR   R9
02          MOV   R10,R10
03          JEQ   SCAN
04          CLR   @KEYADR
05          LI    R1,>1E00
06          BLWP  @VSBW
```

La linea 1 del programma CURSOR, pulisce il registro 9. Le linee 2 e 3 controllano R10 per un valore di 0. Se un registro, una parola, o un byte, è mosso a se stesso, ed il valore del numero è zero, allora il bit di eguaglianza del registro di STATUS, viene fissato a 1. La linea 3 salta all'etichetta SCAN, se R10 è uguale a 0. La linea 4 pulisce gli indirizzi >8374 & >8375. L'istruzione CLR pulisce (fissa a 0 tutti i bit) di una Word di memoria. Gli indirizzo simbolico KEYADR era eguagliato (EQU) a >8374. L'istruzione CLR, pulisce questo byte, ed anche il prossimo (>8375), che è l'indirizzo di KEYVAL.

Con una istruzione, voi avete specificato che volete esplorare l'intera tastiera, e pulire qualsiasi valore precedentemente battuto. La linea 5, carica R1 con il codice del carattere del simbolo del cursore (>1E). Ed infine la linea 6 scrive il simbolo del cursore all'indirizzo di schermo da voi specificato in R0, prima di diramarvi a CURSOR.

```
07          MOV   R0,R8
08          A     R8,R10
09          MOV   R8,R7
```

```
10 SCAN    CLR   @STATUS
```

La linea 7 salva l'indirizzo iniziale del cursore in R8. La linea 8 aggiunge l'indirizzo iniziale di R8 alla lunghezza del valore in R10, per determinare l'indirizzo massimo del cursore. La linea 9 muove l'indirizzo iniziale del cursore in R7, che sarà usato come un accumulatore dei valori degli indirizzi del cursore. La linea 10 pulisce il GPL STATUS byte. Voi dovrete avere il byte di STATUS con tutti zeri, allo scopo di rilevare qualsiasi movimento dei tasti.

```
11         BLWP  @KSCAN
12         CB    @ANYKEY,@STATUS
13         JNE   SCAN
14         MOV   R10,R10
15         JNE   ENTCHK
16         RTN
```

La linea 11 chiama l'utilità di esplorazione della tastiera. La linea 12 usa l'istruzione CB per confrontare il valore del GPL STATUS byte con >20. Questo è il valore che sarà presente nel GPL STATUS byte se un qualsiasi tasto è stato premuto. Alla linea 13, l'istruzione JNE (salta se non uguale) completa il confronto, ritornando all'etichetta SCAN se nessun tasto è stato premuto. La linea 14 controlla ancora R10 con lo zero , muovendo a se stesso, e se R10 non è uguale a zero, allora esso salta all'etichetta ENTCHK. Se invece R10 è uguale a zero, la linea 16 ritorna (B*R11) al programma di chiamata.

```
17 ENTCHK  CB    @ENTERV,@KEYVAL
18         JEQ   ENTER
19         CB    @LEFTV,@KEYVAL
20         JEQ   LEFT
21         CB    @RITEV,@KEYVAL
22         JEQ   RITE
23         C     R7,R10
24         JEQ   SCAN
25         MOV   R7,R0
26         MOVB  @KEYVAL,R1
27         MOVB  @KEYVAL,R9
28         BLWP  @VSBW
29         INC   R7
```

CURSOR salta alla linea 17 se qualche tasto è stato premuto e R10 non è uguale a zero. La prima cosa da fare è di determinare se qualche tasto speciale è stato premuto. Se l'utente ha premuto ENTER, allora esso ha finito di inserire dati. Il valore a >8375, quando il tasto ENTER è premuto, è di >0D. Se l'utente desidera correggere quanto battuto, egli può premere le frecce destra e sinistra (FCTN D, e FCTN S). Il valore del tasto della freccia sinistra è >08, mentre per la freccia destra è >09. Le linee da 17 a 22 controlla no queste condizioni. La linea 23 usa l'istruzione Compare (confronta), per confrontare l'accumulatore dell'indirizzo del cursore (R7) col massimo indirizzo del cursore (R10). Se essi sono uguali, allora la massima lunghezza permessa dei dati è già stata raggiunta. Quando questo è vero, non sono più accettati altri dati dal CURSOR, e i soli valori dei tasti battuti che CURSOR accetterà, sono ENTER e la freccia sinistra. Se il massimo dei valori permessi non è ancora stato raggiunto, le linee 25,26 e 27 accettano i dati dai tasti. La linea 25

copia l'indirizzo nel quale il dato sarà visualizzato da R7 in R0. La linea 26 muove il valore del tasto battuto nel byte sinistro di R1, mentre la linea 27 salva il tasto battuto in R9. La linea 28 scrive il codice del carattere sullo schermo. La linea 29 incrementa R7, che è il nuovo indirizzo del simbolo del cursore.

```
30 CURPUT  MOV    R7,R0
31         LI     R1,>1E00
32         BLWP   @VSBW
33         B      @SCAN
```

La linea 30 mette il nuovo indirizzo che il cursore occupa sullo schermo, in R0. Le linee 31 e 32 scrivono il simbolo del cursore sullo schermo. L'effetto visibile è che il cursore è stato mosso di uno spazio a destra, e il carattere battuto appare nella posizione precedentemente occupata dal cursore. La linea 33 si dirama all'etichetta SCAN, e ripete l'intero processo, formando un ciclo.

```
34 LEFT    C      R7,R8
35         JEQ    SCAN
36         MOV    R7,R0
37         LI     R1,>2000
38         BLWP   @VSBW
39         DEC    R7
40         JMP    CURPUT
41 RITE    C      R7,R10
42         JEQ    SCAN
43         MOV    R7,R0
44         LI     R1,>2000
45         BLWP   @VSBW
46         INC    R7
47         JMP    CURPUT
48 ENTER   LI     R1,>2000
49         MOV    R7,R0
50         BLWP   @VSBW
51         S      R8,R7
52         RT
```

Le linee da 34 a 52 dettagliano le azioni da prendere quando uno dei tasti speciali è premuto. LEFT muove il cursore a sinistra, e riempie il campo lasciato vuoto con un carattere spazio. La linea 34 controlla per vedere se l'indirizzo attuale del cursore (R7) è già al valore minimo (valore iniziale dell'indirizzo). RITE fa giusto il contrario di LEFT.

ENTER è l'etichetta a cui il programma salta quando egli ha determinato che l'utente del computer ha premuto Enter, segnalando la fine dell'ingresso dei dati. Le linee 48, 49, e 50 rimuovono il cursore dall'ultima posizione dello schermo. La linea 51 usa l'istruzione di Word SUBTRACT "S" (sottrai). I contenuti di R8 sono sottratti dai contenuti di R7, ed il risultato viene messo in R7. Questa azione sottrae l'indirizzo iniziale del cursore in R8, dall'ultimo indirizzo dello stesso, in R7. La differenza tra i due è la lunghezza attuale del dato che è stato battuto. L'istruzione RT alla linea 52 ritorna al programma di chiamata.

Supponiamo che una particolare applicazione richieda a chi usa il computer il

nome e cognome. La massima lunghezza del dato che sarà accettata è stata determinata in 30 lettere, (30 bytes). E anche stato determinato che il dato sarà accettato alla riga 10, e colonna 1. Qui sotto vi sono le parti del programma che vi suggeriscono e accettano questi dati.

```
     PROMPT  TEXT  'ENTER FULL NAME '


             LI     R0,256     Presenta il messaggio di prompt alla riga 9,
                               e colonna 1
             LI     R1 PROMPT
             LI     R2,16
             BLWP   @VMBW
             LI     R0,288     Riga 10 e colonna 1
             LI     R10,30     Lunghezza del dato
             BL     @CURSOR    Ottiene il dato
```

A questo punto il nome inserito è visualizzato sullo schermo, iniziando alla riga 10 e colonna 1, e risiede in VDP RAM agli indirizzi da 288 a 317, purchè la lunghezza attuale del dato sia 30. Per fare uso di questi dati nel programma, voi dovrete ottenerlo dal VDP RAM. Per realizzare questo, voi necessiterete della routine VDP MULTIPLE BYTE READ "VMBR" (Lettura di più byte nel VDP). In alternativa voi potreste anche usare VDP SINGLE BYTE READ "VSBR" (lettura di un solo byte in VDP).

Queste routines operano in maniera molto simile a VMBW e VSBW, la sola differenza è la direzione in cui il dato viene mosso. Una lettura muove il dato dall'esterno del programma al programma stesso. Una scrittura muove invece il dato dall'interno del programma a qualche punto all'esterno del programma stesso, come il VDP RAM. Gli stessi registri sono usati per gli stessi parametri. R0 è usato per l'indirizzo in VDP RAM. R1 e R2 regolano l'indirizzo in CPU RAM. Usando VSBR e VMBR con l'E/A deve essere inclusa una direttiva REF.

```
     NAME  BSS 30   Mette da parte temporaneamente memoria per il nome.


     LI     R0,288    Carica R0 con loindirizzo in VDP RAM
     LI     R1,NAME   Carica R1 con l'indirizzo in CPU RAM
     LI     R2,30     Carica R2 con la lunghezza del dato.
                      (presumendo che esso sia lungo 30 bytes)
     BLWP   @VMBR   Esegue un VMBR
```

Ricordate che tutti i dati appaiono uguali al computer, infatti ogni cosa è rappresentata come un'espressione binaria. Quando voi programmate in linguaggio ASSEMBLY, siete voi che dovete decidere come il dato deve essere interpretato. Se, per esempio, un byte di memoria contiene il valore di >41, voi dovete decidere se indica il codice ASCII della lettera "A", o deve essere trattato come valore puramente numerico, cioè 65.

INPUT X in BASIC vi permetterà di inserire solo una stringa numerica, ogni altra cosa viene rifiutata. Nel linguaggio ASSEMBLY voi dovete provvedere a esaminare ogni ingresso del dato, per vedere se esso è numerico, e rifiutarlo se non lo è. Poichè la routine del CURSOR accetta dati di lunghezza variabile, voi dovrete anche decidere se la lunghezza variabile è permessa, o i dati devono essere di lunghezza fissa.

Provate a richiamare una stringa numerica, da utente a tastiera. Per semplificare questo esempio, è richiesto che il numero sia esattamente di quattro cifre, numero intero, o tutti zeri.

```
PROMPT  TEXT 'ENTER A 4 DIGIT NUMBER'  Definisce il prompt del
                                       messaggio.
NMTEST  DATA >3039        L'etichetta "NMTEST" contiene >30, per il
                          codice ASCII di "0", e >39 per "9"
NUMBER  BSS  4            Riserva memoria per i numeri.


GETNUM  LI   R0,256       Indirizzo dello schermo per il prompt; Riga 9
                          e Colonna 1
        LI   R1,PROMPT    Indirizzo del prompt.
        LI   R2,22        Lunghezza del prompt
        BLWP @VMBW        Visualizza il prompt
        LI   R0,288       Indirizzo dello schermo per la risposta al
                          prompt
        LI   R10,4        Massima lunghezza della risposta.
        BL   @CURSOR      Ottiene la risposta
        C    R7,4         Se la lunghezza della risposta non è
                          4, ripete il prompt, e prova ancora.
        JNE  GETNUM
        LI   R0,288       Indirizzo della risposta.
        LI   R1,NUMBER    Dove mettere la risposta.
        LI   R2,4         Lunghezza della risposta.
        BLWP @VMBR        Legge la risposta dal VDP RAM in CPU RAM
        CLR  R3           Fissa R3 a zero.
TEST    CB   @NUMBER(R3),@NMTEST Confronta il byte di "NUMBER", più
                                 il valore di R3 con il byte
                                 all'indirizzo "NMTEST" la prima
                                 volta per R3=0, così, NUMBER + 0.
                                 Il byte a NMTEST = >30, o "0". Se
                                 il byte a NUMBER + R3 è minore di
                                 >30, egli non può essere un numero
                                 ASCII valido.
        JLT  GETNUM       Vai a GETNUM e prova ancora.
        CB   @NUMBER(R3),@NMTEST+1 Confronta il byte a NUMBER+R3
                                   con il byte a NMTEST+1. Il byte
                                   a NMTEST+1 è = a >39, o "9". Se
                                   il a NUMBER+R3 è maggiore di
                                   >39, egli non può essere un
                                   numero valido.
        JGT  GETNUM
        INC  R3           Aggiunge 1 a R3
        CI   R3,R7        Confronta R3 con R7 (R7 contiene 4)
                          JNE TEST Se R3 non è uguale a R7, Torna
                          indietro ed esegui il ciclo TEST ancora.
```

Se il controllo per la lunghezza di 4 viene sostituito con un controllo per una lunghezza di 0, allora queste istruzioni vi permetteranno una lunghezza variabile dei dati. Esempio:

```
     MOV   R7,R7
     JEQ   GETNUM
```

Adesso una stringa di 4 cifre è stata recuperata. I valori attuali della stringa sono codici ASCII di numeri. La sequenza dei simboli numerici rappresentano un numero decimale. Se voi volete usare il valore di questa risposta per qualsiasi tipo di aritmetica ovunque nel vostro programma, essa deve essere convertita in un valore binario. Qui vi è una sequenza delle istruzioni in linguaggio ASSEMBLY del TMS9900 che fa proprio questo.

Questa routine lavorerà solo per numeri decimali non superiori a 65536, il valore massimo per una Word di memoria. Per usare questa routine in un programma, mettete la stringa di numeri in NUMBER, e la lunghezza della stringa in R4, ed eseguite una BL @CONVRT. Il risultato sarà in R5, al completamento della routine, nel formato intero binario. Se il numero da essere convertito è troppo grande, R5 sarà messo a zero. Questa routine presume che voi stiate passando ad essa una stringa numerica valida. Perciò voi dovete controllare i simboli numerici ASCII, prima di eseguire questa routine, allo scopo di ottenere un risultato pienamente significativo.

```
         DTEN   DATA >000A

         NUMBER BSS  6


     01  CONVRT CLR  R0
     02         CLR  R1
     03         CLR  R3
     04         CLR  R5
     05  MOVN   DEC  R4
     06         MOVB @NUMBER(R4),R2
     07         SRL  R2,8
     08         AI   R2,->30
     09         MOV  R0,R0
     10         JNE  EXP
     11         LI   R0,1
     12         JMP  ACCUM
     13  EXP    MPY  @DTEN,R0
     14         MOV  R1,R0
     15         MPY  R1,R2
     16         MOV  R3,R2
     17  ACCUM  A    R2,R5
     18         JNO  NEXT
     19         CLR  R5
     20         RT
     21  NEXT   MOV  R4,R4
     22         JNE  MOVN
     23         RT
```

Ecco come questa routine esegue la conversione dal codice ASCII, a quello BINARIO, ... e anche qualche nuova istruzione Per aiutarvi a comprendere la logica della routine, presumete che il numero da essere convertito sia il decimale 234. Internamente il valore all'indirizzo "NUMBER" può essere rappresentato da una serie di numeri HEX, ciascuno dei quali rappresenta un byte. R4 contiene il valore 3, la

lunghezza della stringa.

Le linee da 1 a 4 puliscono i registri che saranno usati nella routine. La linea 5 DECrementa (cioè, sottrae 1 da...) R4. R4 contiene la lunghezza della stringa. La linea 6 accede alla cifra più bassa usando l'indirizzo di base di NUMBER, più il proprio spiazzamento. La stringa è composta di 3 cifre, e il valore dello spiazzamento delle cifre, che va dai valori più alti a quelli più bassi, sono; 0, 1, 2 (per 234, NUMBER + 0 = >32, = 50 ASCII per il numero 2, NUMBER + 1 = >33, = 51 ASCII per il numero 3, e infine NUMBER + 2 = >34, = 52 ASCII per il numero 4). Il valore dello spiazzamento per l'ultimo byte in una data serie di byte, è SEMPRE UGUALE AL NUMERO DI BYTE MENO 1, in questo caso 3-1=2. "MOVN" è l'etichetta che sarà usata per creare un ciclo.

La prima volta attraverso il cicl, la linea 6 muove il byte di valore inferiore , le unità nel numero 234, cioè 4 (NUMBER + il valore in R4, = NUMBER + 2) a R2. R2 ora contiene >3400. La linea 7 esegue un SHIFT RIGT LOGICAL "SRL" (Spostamento logico a destra) su R2 di 8 posizioni a destra. R2 adesso contiene >0034. La linea 8 usa l'istruzione ADD IMMEDIATE "AI" (Addiziona immediatamente), per togliere la maschera di >30. Usando un valore di ->30, l'effetto sull'istruzione AI, è quello di una sottrazione.

Il registro 2 ora contiene >0004. Poichè la sequenza dei numeri è decimale, voi dovete moltiplicare ogni cifra della potenza di dieci che corrisponda alla stessa posizione in sequenza. La cifra nella posizione più bassa di un numero decimale rappresenta l'unità. La prima cifra che avete estratto moltiplicata per 1 (unità) dovrebbe essere uguale a se stessa. Dunque, la cifra più bassa può essere usata come è. La linea 9 muove R0 a se stesso, cosi che la linea 10 può controllare R0 per un valore di zero. JUMP if NOT EQUAL "JNE" (salta se non uguale). La prima volta R0 è uguale a 0, e l'istruzione JNE della linea 10 non ha effetto. La linea 11 Carica Immediatamente "LI" R0 con il valore di 1. La linea 12 esegue un salto incondizionato (JMP) all'etichetta ACCUM.

Continuiamo con la linea 17 ACCUM, che addiziona i contenuti di R2 e R5. Il valore in R5 (che è stato usato come accumulatore per questa routine) potrebbe diventare troppo grande, e riempire R5, così il bit di traboccamento (OV) verrà fissato a 1 nel registro di status (ST). La linea 18 controlla questa condizione con l'istruzione JUMP if NO OVERFLOW "JNO" (salta se non c'è traboccamento). Finchè il bit di traboccamento non è fissato a 1, l'istruzione logica continua all'etichetta NEXT. Se il bit di traboccamento è messo a 1, allora le prossime due istruzioni puliscono R5, e ritornano al programma di chiamata. Alla linea 21, NEXT muove R4 a se stesso. La linea 22 controlla R4 per un valore di zero, esaminando il bit di uguaglianza nel registro di stato (ST). Se R4 è uguale a zero a questo punto della subroutine logica, il suo compito è fatto, e la linea 23 ritorna all'indirizzo del programma chiamante. la prima volta R4 è uguale a 2, e la routine salta all'etichetta MOVN.

Il secondo passaggio per la routine prende un corso leggermente diverso. R4 viene decrementato di 1. La linea 6 muove il prossimo byte del numero (>33) a R2. R2 viene spostato di 8 posizioni a destra, e diventa >0033, e la maschera di >30 è tolta, dando >0003. R0 ora contiene 1, così la linea 10 scatta un salto all'etichetta EXP alla linea 13.

La linea 13 usa l'istruzione MULTIPLY "MPY" (moltiplica), per calcolare la

potenza di 10 che corrisponde alla posizione relativa della cifra decimale. L'istruzione MPY moltiplica il primo e il secondo operando (che deve essere un registro), e usa due registri successivi come fa l'istruzione DIVide. Come DIVide l'uso di un secondo registro è implicito, significando che il registro addizionale non è specificato in nessuna parte dell'istruzione. Nell'esempio, DTEN viene moltiplicato dai contenuti di R0, e il risultato finisce in R1 perchè R1 è il prossimo registro dopo R0. Ora esaminate il contenuto dei registri implicati:

PRIMA DI MPY	DTEN = >000A	R0 = >0001	R1 = >0000
DOPO MPY	" = >000A	R0 = >0000	R1 = >000A

Se il risultato di una moltiplicazione dovesse essere più grande di una Word, egli resterà nel registro nominato. [5]In questo esempio il registro 0 contiene i bit più importanti, e il registro 1 contiene i bit meno importanti. L'istruzione MPY non ha effetto sul registro di stato (ST).

La linea 14 muove il risultato di 10, che è in R1, in R0. Ciò è fatto per preparare R0 al prossimo ciclo. Ad ogni passaggio per il ciclo, R0 sarà moltiplicato per 10. In questa maniera, i contenuti di R1 saranno uguali alla potenza di 10 necessaria per ogni posizione decimale (1, 10, 100, 1000 ecc.). Ricordate che un movimento copia soltanto i contenuti di una locazione ad un'altra. I contenuti di R1 sono sempre intatti. Alla linea 15, il valore in R2, che è il numero da essere convertito (>0003) viene moltiplicato per il valore in R1 (la potenza del dieci per la posizione di questo numero). Ecco l'effetto della moltiplicazione della linea 15:

PRIMA DI MPY	R1 = >000A	R2 = >0003	R3 = >0000
DOPO MPY	R1 = >000A	R2 = >0000	R3 = >001E

Il risultato finisce in R3, il primo registro disponibile dopo R2, che era il registro nominato nel secondo operando dell istruzione MPY. La linea 16, muove il risultato in R2. Questo viene fatto perchè la linea 17, ACCUM, si aspetta che il valore da essere addizionato a R5 debba essere in R2.

Il ciclo viene ripetuto un'altra volta per estrarre la cifra più alta. Questa cifra rappresenta le unità di 10 o 100. Il numero estratto è due. Moltiplicato per 100, diventa uguale a 200. Addizionato a R5 da ACCUM, porta il valore totale di R5 a >00EA, o 234 Dec. Il valore in R5 può ora essere usato per qualsiasi calcolo aritmetico di cui avrete bisogno.

Voi non avrete bisogno di usare una routine estesa come CURSOR, nel vostro programma interattivo con l'ingresso da tastiera, o per fare buon uso di una 'utilità come KSCAN. Nei primi due programmi esempio, l'indirizzo di ritorno era stato alterato con lo scopo di permettervi di osservare i risultati del programma stesso. Adesso che conoscete qualche cosa su KSCAN, c'è qui un modo migliore per terminare un programma. L'indirizzo di ritorno è inalterato, e, allo scopo di creare una pausa, KSCAN viene usata per rivelare la pressione di un qualsiasi tasto. Il programma attenderà finchè non sia premuto un tasto, e poi termina. Ecco una parte di programma che fa proprio questo:

```
EOJ   CLR  @STATUS    Pulisce il GPL STATUS byte
SCAN  BLWP @KSCAN     Esegue la scansione della tastiera
      MOVB @STATUS,@STATUS  Confronta il GPL STATUS byte con >00
      JEQ  SCAN       Se nessun tasto è stato premuto, esplora ancora
      MOV  @SAVRTN,R11 Muove l'indirizzo di ritorno a R11
      CLR  @STATUS    Pulisce il GPL STATUS byte
      RT              RETURN per mezzo di R11
```

Un'altra applicazione di KSCAN è la rivelazione della pressione dei tasti speciali, come CLEAR o QUIT. Ricordate che quando il vostro programma nel linguaggio ASSEMBLY del TMS9900, sta girando,egli è sotto il completo controllo del computer. Allo scopo di poter uscire dal programma, il programma stesso deve provvedere a rivelare l'uso di un simile comando. Poichè questi tipi di tasti non rappresentano dati visualizzabili, non c'è bisogno di una routine come quella del CURSOR. Dipendentemente da come programmate le azioni da essere prese per i vari valori di controllo, il vostro computer può rispondere conformemente a questi comandi. Ecco una parte di programma per rivelare la pressione del tasto QUIT (FCTN=):

```
QUITV  BYTE   >05


SCAN   CLR    @STATUS   Pulisce il GPL STATUS byte
       BLWP   @KSCAN    Esegue la scansione della tastiera
       MOVB   @STATUS,@STATUS  Vede se un tasto è stato premuto
       JEQ    SCAN      Altrimenti esplora di nuovo
       CB     @QUITV,@KEYVAL  Vede se il tasto QUIT è stato premuto
       JEQ    ABORT     Se lo era, fine del lavoro
       RT               Altrimenti RETURN
```

IMPORTANTE: Prima di uscire da un programma con QUIT, assicuratevi di chiudere qualsiasi file aperto.

La risposta normale alla pressione del tasto QUIT (FCTN =), quando si opera in TI BASIC, EX/BASIC, e la maggior parte delle utilità, è per il computer, il ritorno al titolo principale di schermo, (Quello a barre colorate). Ecco quì sotto la parte del programma che fa questo:

```
GPLWS  EQU    >83E0


ABORT  LIMI   2      Attiva le interruzioni
       LWPI   GPLWS  Carica i registri workspace GPL
       BLWP   @>0000 Si dirama al vettore >0000
```

L'istruzione LIMI sta per "LOAD INTERRUPT MASK IMMEDIATE" (Carica immediatamente la maschera d'interruzione), ed è usata per attivare/disattivare le interuzioni. LIMI 0 (Interruzioni disattivate) è lo stato normale del computer. Questa istruzione mette i quattro bit meno importanti del contenuto dell'operando immediato nella maschera d'interruzione del registro di STATUS.

Senza interruzioni, la CPU processa un'istruzione o un dato, uno dopo l'altra. Questa sequenza del processo si ripete ad ogni impulso costante. Certe operazioni, richiedono che voi interrompiate questo processo regolarmente, normalmente per

permettere le diverse velocità dei microprocessori che compongono un computer. LIMI 2 attiva le interruzioni ai livelli 0, 1, e 2. Poichè la diramazione per >0000 ritorna al titolo principale dello schermo, per mezzo della routine GPL residente, è necessario avere il registro WP puntato al Workspace usato dal GPL.

Ecco qui un a tabella che da i valori dei tasti numerici premuti, in combinazione con il tasto FCTN.

COMBINAZIONI	NOME	VALORE DEL TASTO PREMUTO DECIMALE	HEX
FCTN 1	DELETE	03	>03
FCTN 2	INSERT	04	>04
FCTN 3	ERASE	07	>07
FCTN 4	CLEAR	02	>02
FCTN 5	BEGIN	14	>0E
FCTN 6	PROCEED	12	>0C
FCTN 7	AID	01	>01
FCTN 8	REDO	06	>06
FCTN 9	BACK	15	>0F
FCTN 0		188	>BC
FCTN =	QUIT	05	>05

Quello che segue è un programma d'esempio che dimostra alcuni dei principi della routine KSCAN. La grafica per il bordo dello schermo, e per il prompt, usano la stessa logica del programma esempio del capitolo sesto. Quando esso gira, qualsiasi tasto che voi premete farà visualizzare il carattere battuto, (purchè esso risulti visualizzabile), ed il valore decimale del tasto o combinazioni di tasti. Il prossimo prompt accetterà solo REDO o ESCAPE. REDO ripete l'intera sequenza, mentre ESCAPE vi ritornerà al titolo principale dello schermo. Voi potete usare questo programma per trovare il valore di ogni tasto, o combinazione di tasti premuti.

La routine FIGUR usa la stessa logica del primo programma esempio, quello per convertire una cifra binaria nel corrispondente codice ASCII visualizzabile. FIGUR usa un ciclo per permettere ad esso di processare qualsiasi numero idoneo in un solo registro. FIGUR tratta la convesrione e visualizza il valore.

```
*PROGRAMMA ESEMPIO DI INGRESSO DA TASTIERA
*VERSIONE E/A

        DEF   GO
        REF   VWTR,VSBW,VMBW,KSCAN
WR      BSS   >20                       Mette da parte spazio per WS
STATUS  EQU   >837C                     STATUS byte GPL
KEYADR  EQU   >8374                     Indirizzo della tastiera
KEYVAL  EQU   >8375                     Indirizzo del tasto premuto
DTEN    DATA  >A                        Decimale 10
BORDER  DATA  >FFFF,>2020,>2020,>2020 Definisce il pattern del bordo
        DATA  >2020,>2020,>2020,>2020
        DATA  >2020,>2020,>2020,>2020
```

```
         DATA  >2020,>2020,>2020,>FFFF
MSG1     TEXT  '** PRESS ANY KEY    **'  Definisce il primo prompt
MSG2     TEXT  '* KEYSTROCHE VALUE IS *' Definisce il secondo prompt
MSG3     TEXT  '* PRESS REDO/ESCAPE   *' Definisce il terzo prompt
REDOV    BYTE  >06                       Valore per "REDO"
ESCPV    BYTE  >0F                       Valore tasto "ESCAPE"
SAV11    BSS   2
GO       MOV   R11,@SAV11               Salva l'indirizzo di ritorno
         LWPI  WR                        Carica il LWPI
         LI    R0,070D                   Fissa il colore di sfondo
*                                        del bordo come magenta
         BLWP  @VWTR
         LI    R0,>039F                  Fissa il caratter del bordo
*                                        come magenta
         LI    R1,>DD00
         BLWP  @VSBW
         LI    R0,>380                   Caratteri bianchi su nero
         LI    R1,>1F00
CLOOP    BLWP  @VSBW
         CI    R0,>039E
         JEQ   BPUT
         INC   R0
         JMP   CLOOP
BPUT     LI    R0,0                      Carica i caratteri del bordo
         LI    R1,BORDER
         LI    R2,32
BLOOP    BLWP  @VMBW
         CI    R0,736
         JEQ   EXIT
         AI    R0,32
         JMP   BLOOP
EXIT     LI    R0,261                    Presenta il primo prompt
         LI    R1,MSG1
         LI    R2,22
         BLWP  @VMBW
         CLR   @KEYADR                   Pulisce KEYADR e KEYVAL
SCAN1    CLR   @STATUS                   Pulisce il GPL STATUS byte
         BLWP  @KSCAN                    Esegue KSCAN
         MOVB  @STATUS,@STATUS          Vede se un qualsiasi tasto
*                                        è stato premuto
         JEQ   SCAN1                     Altrimenti esplora ancora
         LI    R0,325                    Presenta il secondo prompt
         LI    R1,MSG2
         BLWP  @VMBW
         LI    R0,395                    Visualizza i caratteri dei
*                                        tasti
         MOVB  @KEYVAL,R1
         BLWP  @VSBW
         MOVB  @KEYVAL,R4
         SRL   R4,8
         LI    R3,404                    Visualizza il valore
*                                        decimale dei caratteri
```

```
         LI     R0,406
         BL     @FIGUR
         LI     R0,485                     Presenta il terzo prompt
         LI     R1,MSG3
         LI     R2,22
         BLWP   @VMBW
SCAN2    CLR    @STATUS                    Esegue ancora KSCAN
         BLWP   @KSCAN
         MOVB   @STATUS,@STATUS
         JEQ    SCAN2
         CB     @KEYVAL,@ESCPV             ESCAPE O REDO premuti?
         JEQ    ESCAP                       Se ESCAPE, salta a ESCAP
         CB     @KEYVAL,@REDOV
         JNE    SCAN2
         B      @BPUT                      Se REDO salta a BPUT
FIGUR    MOV    R4,R5                       Routine per convertire i
*                                           numeri interni in numeri
*                                           decimali visualizzabili
*                                           Ingresso: R4=valore, R3=
*                                           primo indirizzo sullo
*                                           schermo del risultato
*                                           R0=ultimo indirizzo
*                                           sullo schermo
         CLR    R4
         DIV    @DTEN,R4
         AI     R5,>30
         SLA    R5,8
         MOV    R5,R1
         BLWP   @VSBW
         DEC    R0
         C      R0,R3
         JHE    FIGUR
         RT
ESCAP    CLR    @STATUS                     Pulisce lo STATUS byte
         MOV    @SAV11,R11                 Ritorna
         RT
         END
```

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sull'ingresso dalla tastiera
Sezione 10-2 pag.164
Sezione 16-2 pag.250
Dalla sezione 16-3 pag. 264, alla sezione 16-3-1 pag.264
Dalla sezione 24-11 pag.440, alla sezione 24-11-3 pag.442

GUARDARE QUESTI TERMINI SUL GLOSSARIO
ASCII
Console
Bit della maschera d'interruzione

CAPITOLO NONO

TRATTAMENTO DEI FILE

La creazione, la lettura, e l'aggiornamento dei dati di un FILE, attraverso il linguaggio ASSEMBLY del TMS9900, è un'altra importante funzione che coinvolge il VDP RAM. Le specificazioni del FILE che descrivono la lunghezza del record, la lunghezza del formato, formato dei file e dati, e modi di operazione, sono accumulati in un blocco di memoria conosciuto come "PERIPHERAL ACCESS BLOCK" (PAB) (Blocco d'accesso periferiche)

Il PAB dettaglia tutte le informazioni richieste dal computer per riconoscere ed accedere ai file che volete. La lettura, scrittura e aggiornamento dei dati del file, è trattato da una routine residente chiamata "DEVICEs SERVICE ROUTINE" (DSR) (Routine del dispositivo di servizio).

Il trattamento dei file è implementato attraverso il linguaggio ASSEMBLY del TMS9900 manipolando i dati dentro il PAB per un file, e creando i dati disponibili nel PAB, per il corretto DSR.

Quando programmate in TI BASIC, voi fornite tutti i parametri sopra un file, nell'istruzione OPEN, la quale apriva il file e specificava come dovevano essere usati INPUT, OUTPUT, UPDATE, o APPEND. Il file era poi accessibile dentro al programma dalle istruzioni INPUT o PRINT. Quando poi avevate finito con il file , era necessario chiuderlo. Queste quattro funzioni del file, che lo definiscono, lo aprono, lo leggono o scrivono, ed infine lo chiudono, sono sempre richieste da qualsiasi linguaggio di programmazione. Il linguaggio ASSEMBLY del TMS9900, richiede ancora qualche linea di codice in più, per ottenere lo stesso risultato del TI BASIC, ma il trattamento dei file non è generalmente molto complesso.

Il primo passo è quello di definire le caratteristiche del file al computer. I byte dei dati che creano un PAB, definiscono i parametri chiave del file. Il numero attuale dei byte che creano un PAB è variabile, dipendendo dal nome del dispositivo/file selezionato. Il primo byte (il byte zero) del PAB istruisce il DSR sul tipo di operazione che voi desiderate eseguire; (Open, Read, Write, Close, etc.) (aprire, leggere, scrivere, chiudere, etc.). Il fissaggio iniziale di questo byte per la maggior parte dei file dovrebbe essere >00, o OPEN.

Ecco gli OP-CODE (codici operativi) disponibili per il byte zero del PAB.

VALORE	OPERAZIONE
>00	OPEN
>01	CLOSE
>02	READ
>03	WRITE
>04	RESTORE/REWIND
>05	LOAD
>06	SAVE
>07	DELETE FILE
>09	STATUS

L'OP-CODE >08 (Scratch Record) (cancella record) non è generalmente usato dal TI-99/4A,perchè il disk controller non permette ad un record di essere cancellato.

Il prossimo byte del PAB (il byte 1)controlla parecchie funzioni, dipendendo da

quali bit sono messi ON o OFF (accesi o spenti, 1 o 0), esso definisce il modo di apertura (OPEN) (input,output,update), tipo di record (fixed o variable) (di lunghezza fissa o variabile), il tipo di dati contenuti (Display, o internal) (formato visualizzabile o interno), e se il file deve essere sequenziale o relative (ad accesso casuale). Tutti questi parametri sono definiti da varie combinazioni dei bit da 3 a 7 del byte 1. I bit 0, 1, 2 sono usati per riportare le varie condizioni di errore che possono capitare. Il fissaggio iniziale dei bit 0, 1, e 2 dovrebbe essere sempre zero, (nessun errore).

Ecco un sommario dei valori del byte 1 del PAB, ed i loro significati.

PAB BYTE 1 VALORI E SIGNIFICATO

FILE RELATIVI - (Tutti i file relativi sono di lunghezza FIXED

>01	UPDATE, DISPLAY
>03	OUTPUT, DISPLAY
>05	INPUT, DISPLAY
>09	UPDATE, INTERNAL
>0B	OUTPUT, INTERNAL
>0D	INPUT, INTERNAL

FILE SEQUENZIALI

>02	OUTPUT, FIXED, DISPLAY
>04	INPUT, FIXED, DISPLAY
>06	APPEND, FIXED, DISPLAY
>0A	OUTPUT, FIXED, INTERNAL
>0C	INPUT, FIXED, INTERNAL
>0E	APPEND, FIXED, INTERNAL
>12	OUTPUT, VARIABLE, DISPLAY
>14	INPUT, VARIABLE, DISPLAY
>16	APPEND, VARIABLE, DISPLAY
>1A	OUTPUT, VARIABLE, INTERNAL
>1C	INPUT, VARIABLE, INTERNAL
>1E	APPEND, VARIABLE, INTERNAL

I byte 2 e 3 (una word) contengono l'indirizzo in VDP RAM che deve essere usato come un buffer (spazio di memoria temporaneo) per ogni record che viene letto o scritto. Il byte 4 definisce la lunghezza logica del record in byte. Per una lunghezza variabile questo valore indica la massima lunghezza del record. Il valore più grande che può essere definito usando un byte è >FF, o 255. Il byte 5 definisce il numero di byte da scrivere con un'operazione di scrittura, o il numero di byte da leggere, con un'operazione di lettura.

Per un record di lunghezza FIXED (fissa), i byte 4 e 5 del PAB dovranno essere fissati sempre uguali sia in scrittura che in lettura. Per record di lunghezza variabile, il byte 5 del PAB può essere esaminato per determinare la lunghezza

del record in lettura, e può essere cambiato dinamicamente per ogni scrittura. Il valore del byte 5 del PAB non può mai essere maggiore della lunghezza logica, o massima del record. I byte 6 e 7 del PAB (una word) vengono usati solo per file relativi (ad accesso casuale). Questa word contiene il numero del record relativo a cui accedere. Il bit più importante di questa word è ignorato, così che la gamma dei

valori possibili parte da zero (il primo record di un file relativo) fino a 32.767. Il byte 8 è usato solo per i file memorizzati su un dispositivo a cassette (registratore).

Il valore in questo byte è il totale dell'offset dello schermo (>60 in BASIC, >00 in ASSEMBLY). Il DSR della cassetta ha bisogno di questo valore per il prompt (messaggio) sullo schermo, che deve visualizzare le operazioni per il registratore. Il byte 9 indica la lunghezza del descrittore del file, che comincia nel byte 10. La descrizione del file può essere di lunghezza variabile.

Qui viene indicato dove mettere il FILE/DEVICE NAME (Nome del dispositivo) che avete scelto per il vostro file ("CS1", "DSK1.", "RS232.BA=300", etc.). poichè la lunghezza di questo input varierà in dipendenza del dispositivo selezionato, il computer necessita di questi dati nel byte 9.

Ecco come dovrebbe essere codificato un PAB in un programma in linguaggio ASSEMBLY del TMS9900.

```
        TEXT  'DSK1.FILE1'
```

Descrittore del file
(Questo descrittore è lungo
>0A (10) caratteri)

I PAB sono codificati nel vostro programma, e poi messi in VDP RAM con una routine come VMBW. Il VDP RAM è usato da tutti i DSR per PAB e spazio buffer. Poichè ci sono tavole e altri dati molto importanti in VDP RAM, solo certe aree saranno usate per il PAB e buffer. Il primo indirizzo libero in VDP RAM normalmente usato per i PAB è >F80. Questo indirizzo effettivamente si sovrappone alla fine della Tavola Descrittrice dei Pattern, ma i codici dei caratteri da >F0 a FF non sono definiti come visualizzabili, cosi questo normalmente non causa problemi.

Lo spazio libero in VDP RAM si estende attraverso l'indirizzo >37D6. Questo rappresenta uno spazio totale considerevole per il PAB e le necessità del buffer. State attenti a usare questo spazio solo per queste funzioni. Dove molti file devono essere usati, deve essere riservato spazio a sufficienza tra i PAB e i buffer per assicurare l'integrità dei dati del file .

Qui ci sono parti di programma che stabiliscono un PAB e un buffer:

```
BUFADR   EQU   >1000  Indirizzo VDP RAM per buffer record
PABADR   EQU   >0F80  Indirizzo VDP RAM per il PAB
PAB      DATA  >0012,BUFADR,>5000,>0000,>0009
```

```
        TEXT  'DSK1.FILE '


        LI    R0,PABADR
        LI    R1,PAB
        LI    R2,20
        BLWP  VMBW
```

Una volta che il PAB ed il buffer per il file sono stati stabiliti, l'effettivo accesso è compiuto per mezzo del DSR. Puntando al PAB (definizione del file) a cui avere accesso, manipolando il byte zero del PAB (operazione), e diramandosi al DSR, il file può essere aperto, letto, scritto, chiuso, etc. Il DSR per tutte le periferiche, eccetto le cassette, è accessibile includendo una REF DSRLNK nel vostro programma, usando l'E/A, oppure una direttiva EQU all'indirizzo >6038, con alcune etichette se usate l'assembler LINEA per LINEA.

La DSR per le cassette è una routine GPL (Graphics Programming Language) (Linguaggio di programmazione grafico) situato in GROM (Graphics Read Only Memory) (memoria di sola lettura grafica). Il trattamento dei file da cassetta, e DSR saranno discussi più tardi. Il puntatore necessario per DSRLNK è l'indirizzo del byte del descrittore del file. Questo valore deve essere messo nella word all'indirizzo >8356. Per l'esempio di cui sopra, allora

```
        LI    R6,PAB+9
        MOV   R6,@>8356
```

DSRLNK viene poi richiesto con una istruzione BLWP. Il DSRLNK ha bisogno anche del valore 8 per completare l'istruzione.

```
        BLWP  @DSRLNK
        DATA  8
```

La stessa istruzione DSRLNK è usata per qualsiasi operazione su qualsiasi file, escluse le cassette. Le reali operazioni eseguite su di un file da ogni BLWP a DSRLNK (open, read, ect.)dipende dal valore nel byte zero del PAB. Il file accessibile dipende dal valore messo a >8356. Il DSRLNK determinerà se le caratteristiche del file corrispondono al dispositivo, e se l'operazione richiesta è compatibile con le caratteristiche del dispositivo/file. Gli errori di questo tipo, come EOF (del disco) (End Of File) (fine del file) e altre condizioni del processo sono rivelate da DSRLNK e riportate nei bit 0, 1, e 2 del byte 1 del PAB. Presumendo che non capitino errori durante il processo del file, DSRLNK tratta tutti gli aspetti del processo, come un'aggiornamento del catalogo entrate per un file su disco.

Il programma nel linguaggio ASSEMBLY del TMS9900, deve controllare le condizioni di errore, e provvedere, come risultato, alle azioni da prendere. Quando capitano degli errori, il bit di uguaglianza nel registro di status (ST) viene messo a 1 (settato). Se non capitano errori, lo stesso bit viene messo a zero (resettato).

Se il dispositivo che avete selezionato è RS232, o PIO, voi dovete salvare gli indirizzi di lettura e scrittura GROM prima di ciascun BLWP @DSRLNK, e ristabilirli in seguito. La DSR per questi dispositivi rendono questi indirizzi indeterminati. Ecco una parte di programma per salvare e ristabilire questi indirizzi.

```
        REF   GRMRA
```

```
        REF    GRMWA

SAVADR  BSS    2

        MOVB   @GRMRA,@SAVADR  Ottiene il primo byte dell'indirizzo
        NOP
        MOVB   @GRMRA,@SAVADR+1 Ottiene il secondo byte
        DEC    @SAVADR          Decrementa l'indirizzo

        BLWP   @DSRLNK         Accede alla routine periferiche
        DATA   8

        MOVB   @SAVADR,@GRMWA  Ristabilisce il primo byte
*                               dell'indirizzo
        NOP
        MOVB   @SAVADR+1,@GRMWA  Ristabilisce il secondo byte
*                                 dell'indirizzo
```

Nell'esempio sopra riportato, la pseudo istruzione "NOP" viene usata per permettere un ritardo di tempo per l'accesso agli indirizzi GROM. NOP non esegue nessuna funzione, ma prende tempo come farebbe una istruzione reale.

Qui c'è un programma esempio di file ad accesso sequenziale. L'input del file contiene record di lunghezza VARIABLE 80, nel formato DISPLAY. Ogni record contiene un primo nome con una lunghezza massima di 14 caratteri, ed un ultimo nome. La lunghezza effettiva del campo dell'ultimo nome (e quindi del record) è variabile, benchè il primo carattere dell'ultimo nome deve cominciare dalla quindicesima posizione. Questo programma legge il nome del file e seleziona il terzo record sul file. Il primo e l'ultimo nome del terzo record saranno poi visualizati.

```
0001          DEF  BEGIN
0002          REF  DSRLNK,VSBW,VMBW,VSBR,VMBR
0003 STATUS   EQU  >837C        Indirizzo dello STATUS byte GPL
0004 POINTR   EQU  >8356        Indirizzo del puntatore al DSR
0005 BUFADR   EQU  >1000        Indirizzo del buffer record in VDP RAM
0006 PABADR   EQU  >F80         Indirizzo VDP RAM per il PAB
0007 READ     BYTE >02          Codice operativo di "READ"
0008 CLOSE    BYTE >01          Codice operativo di "CLOSE"
0009 EOF      DATA >0           Segnalatore (FLAG) di fine del file
0010 PAB      DATA >0014,BUFADR,>5000,>0000,>000A   Dati per il PAB
0011          TEXT 'DSK2.FILE1'
0012 ERRMSG   TEXT 'I/O ERROR='  Messaggio di errore del DSR
0013 CPUBUF   BSS  80           Indirizzo del buffer record in CPU RAM
0014 FNAME    EQU  CPUBUF       Indirizzo del primo nome
0015 LNAME    EQU  CPUBUF+14    Indirizzo dell'ultimo nome
0016 LEN      BSS  2            Lunghezza effettiva del record WorkSp.
0017 RETURN   BSS  2            Salva l'indirizzo dell'area di ritorno
0018 WR       BSS  >20          Registri di lavoro
0019 BEGIN    MOV  R11,@RETURN Salva l'indirizzo di ritorno
0020          LWPI WR           Carica i registri di lavoro
0021          LI   R0,PABADR    Indirizzo in VDP RAM per il PAB
0022          LI   R1,PAB       Indirizzo in CPU RAM per i dati del PAB
0023          LI   R2,20        Lunghezza dei dati
0024          BLWP @VMBW        Scrive il PAB in VDP RAM
0025          BL   @DSR         Apre il file
0026          MOVB @READ,R1     Carica l'OP-CODE di READ in R1
0027          LI   R0,PABADR    Carica l'indirizzo del PAB in R0
0028          BLWP @VSBW        Mette READ nel byte 0 del PAB
0029          CLR  R4           Pulisce il contatore dei record
0030 READF    BL   @DSR         Esegue la routine del DSR
0031          MOV  @EOF,@EOF    Controlla la fine del file
0032          JNE  EOJ          Se EOF andare alla fine del lavoro
0033          INC  R4           Aggiunge 1 al contatore dei record
0034          CI   R4,3         Controlla per il terzo record
0035          JNE  READF        Se non è il terzo, leggi ancora
0036          LI   R0,PABADR+5 Indirizzo del contatore di caratteri
0037          BLWP @VSBR     Legge il contat. nel byte sinist. di R1
0038          SRL  R1,8         Sposta da sinistra a destra
0039          MOV  R1,R2        Muove il valore in R2
0040          MOV  R1,@LEN      Salva il valore dentro  lunghezza
0041          LI   R0,BUFADR    Indirizzo del buffer record in VDP RAM
0042          LI   R1,CPUBUF    Indirizzo del record in CPU RAM
0043          BLWP @VMBR        Ottiene il record da VDP a CPU RAM
0044          LI   R0,290       Indirizzo schermo per il primo nome
0045          LI   R1,FNAME     Indirizzo del primo nome in CPU RAM
0046          LI   R2,14        Lunghezza del campo del primo nome
0047          BLWP @VMBW        Visualizza il primo nome
0048          LI   R0,305       Indirizzo schermo per l'ultimo nome
0049          LI   R1,LNAME     Indirizzo in CPU RAM dell'ult. nome
0050          MOV  @LEN,R2      Muove la lunghezza del record in R2
0051          AI   R2,-14       Sottrae la lunghezza del primo nome, e
     *                          la differenza è la lunghezza
```

```
     *                          dell'ultimo nome
0052          BLWP @VMBW        Visualizza l'ultimo nome
0053          JMP  EOJ          Vai alla fine del lavoro
0054 DSR      LI   R6,PABADR+9  Carica R6 con il descritt. della lung.
0055          MOV  R6,@POINTR   Muove l'indirizzo al Puntatore
0056          BLWP @DSRLNK      Esegue il DSRLNK
0057          DATA 8            Dati necessari per il DSRLNK
0058 *
0059          JEQ  DSRERR       Controlla per errori
0060          RT                Ritorna
0061 DSRERR   INC  @EOF         Fissa l'indicatore EOF
0062          LI   R0,PABADR+1  Indirizzo del byte 1 del PAB
0063          BLWP @VSBR        Legge il byte 1 del PAB in R1
0064          SRL  R1,13        Sposta i 3 bit più alti al posto dei
     *                          più bassi
0065          CI   R1,5         Controlla per il valore di EOF = 5
0066          JNE  IOERR        Se non è EOF allora altro errore
0067          RT                Se è EOF allora ritorna
0068 IOERR    AI   R1,>30       Maschera del codice di errore
0069          SLA  R1,8         Scambia i bit di ordine basso con
     *                          quelli di ordine alto
0070          LI   R0,299       Visualizza il codice di errore
0071          BLWP @VSBW        ...sullo schermo
0072          LI   R0,288       Visualizza il messaggio di errore
0073          LI   R1,ERRMSG
0074          LI   R2,10
0075          BLWP @VMBW
0076 EOJ      MOV  @EOF,@EOF    Se è stato raggiunto EOF, allora
0077          JNE  NOCLOS       Chiuderà il file
0078          MOVB @CLOSE,R1    Muove l'OP-CODE di CLOSE in R1
0079          LI   R0,PABADR    Carica l'indirizzo del PAB
0080          BLWP @VSBW        Scrive l'OP-CODE di CLOSE al PAB 0
0081          BL   @DSR         Chiude il file
0082 NOCLOS   DECT @RETURN      Altera l'indirizzo di ritorno
0083          MOV  @RETURN      Muove l'indirizzo di ritorno in R11
0084          RT                Ritorna
0085          END
```

COMMENTI

La linea 25 apre il file perchè l'OP-CODE nel byte 0 del PAB è originariamente fissato a >00 (L'OP-CODE di "OPEN"). Le linee 26, 27, e 28 scrivono l'OP-CODE per la lettura (READ >02) nel byte 0 del PAB. Una volta che questo è fatto, ogni DSRLNK successivo esegue una lettura e continuerà ad eseguire letture finchè l'OP-CODE non viene cambiato in qualche altro valore. La subroutine DSR dalle linee 54 fino a 60 contengono le istruzioni e i dati che esegue il DSR. La linea 59 usa l'istruzione JEQ (Salta se uguale), per esaminare il bit di uguaglianza nel registro di STATUS (ST). DSRLNK fissa questo bit se ci sono degli errori. EOF è riportato come errore con un valore di 5 nei bit 0, 1, e 2 del byte 1 del PAB

Le linee da 61 a 67 esaminano per EOF. Se EOF è stato raggiunta, allora la parola "EOF" è cambiata da zero a uno. DSRLNK Chiuderà il file per voi se è stato raggiunta EOF. Per questo programma esempio la EOF capiterà solo se il file conteneva

meno di 3 record. Se la condizione riportata è un pò oltre la fine del file, le linee da 68 a 75 visualizzano il codice di errore ed un messaggio.

Le linee da 30 a 53 spiegano dettagliattamente le azioni che leggono attraverso il file finchè il terzo record è trovato. Le linee 36 e 37 estraggono il contatore di caratteri dal byte 5 del PAB. VSBR legge dall'indirizzo in R0 nel byte sinistro di R1. Il valore dal byte 5 del PAB è la lunghezza effettiva del record appena letto. Questo valore è usato insieme con VMBR per ottenere il record dal suo buffer in VDP RAM, in un buffer in cui il programma può accedere direttamente (linee 39/43). Le linee 44/47 ottengono e visualizzano il primo nome. Le linee 48/52 fanno la stessa cosa con l'ultimo nome. Il calcolo alla linea 51 determina l'effettiva lunghezza del campo dell'ultimo nome, sottraendo la lunghezza del campo del primo nome dall'effettiva lunghezza del record che fu trovato nel byte 5 del PAB.

Alla fine del lavoro (linea 76) una decisione deve essere presa perchè in ogni caso il file necessita di essere chiuso. Per chiuderlo l'OP-CODE per "CLOSE" (>01) viene scritto nel byte 0 del PAB e viene eseguito un altro accesso a DSRLNK. Poi il programma segue i precedenti esempi che attendono che venga premuto un tasto per finire. Ecco un programma in TI BASIC che creerà un nome del file simile a quelo usato come input al programma esempio nel linguaggio. ASSEMBLY.

```
100 CALL CLEAR
110 OPEN #2:"DSK2.FILE1",OUTPUT,VARIABLE 80
120 INPUT "INSERIRE UN "E" QUANDO AVETE FINITO":X$
130 IF X$="E" THEN 190
140 INPUT "PRIMO NOME ?":PN$
150 IF LEN(PN$)>14 THEN 140
160 INPUT "ULTIMO NOME ?":UN$
170 PRINT #2:PN$,UN$
180 GOTO 120
190 CLOSE #2
200 END
```

Quì di seguito vi è una visualizzazione esadecimale del file creato dal programma in TI BASIC. Questo è stato prodotto con il dischetto "AIUTO ALLA PROGRAMMAZIONE II" una utility della TEXAS INSTR. Mentre il programma in BASIC limita la dimensione del campo del primo nome ad un massimo di 14 caratteri, non c'è tale limite con le dimensioni dell'ultimo nome. Se voi intendete processare un file creato con il TI BASIC per mezzo dell'ASSEMBLY del TMS9900, è importante che il programma in TI BASIC sia codificato per specificare determinate posizioni di campi, e i campi lunghezza. Oppure come è stato fatto in questo caso, voi potete usare l'utility DUMP dell'Aiuto alla Programmazione II, per svelare come i record del file appariranno nel programma ASSEMBLY.

IL TIPO DI FILE E DISPLAY

IL TIPO DI RECORD E VARIABLE 80

```
13 44 41 56     (.DAV)
49 44 20 20     (ID  )
20 20 20 20     (    )
20 20 20 53     (   S)
54 4F 4E 45     (TONE)
14 4D 41 52     (.MAR)
56 49 4E 20     (VIN )
20 20 20 20     (    )
20 20 20 53     (   S)
50 41 52 4B     (PARK)
53 15 57 41     (S.WA)
59 4E 45 20     (YNE )
20 20 20 20     (    )
20 20 20 20     (    )
4E 45 57 43     (NEWC)
4F 4D 45        (OME )
```

Quando create file con i programmi in ASSEMBLY del TMS 9900, voi dovete designare la disposizione di ogni record. Voi decidete quanto può essere lungo ogni campo, e la sua posizione di inizio nel record. Una delle inefficienze del BASIC è che assegna lo spazio per le stringhe dentro un record in predeterminati blocchi di byte. A meno che voi non codifichiate il programma BASIC per strutturare ogni campo, il TI BASIC segue un algoritmo predederminato per la lunghezza del campo.

Le stringhe alfanumeriche, per esempio, partono sempre con una lunghezza di 14 byte. Nel caso del campo del primo nome, anche se tutti i primi nomi inseriti non sono mai più lungi di 4 o 5 byte, vengono costruite stringhe di 14 byte per ognuno. Togliendo la linea 150, che restringe il campo della lunghezza, se il primo nome inserito ha una lunghezza di 15 byte, allora un blocco addizionale di 14 byte sarà aggiunto al primo. Così sarà creato uno spazio stringa per un blocco di 28 byte, quando sarebbe stato sufficente uno spazio per 15 byte. Questo diventa ancora peggio con stringhe numeriche. Anche se un numero di una sola cifra viene inserito, verrà creato un spazio per una stringa di 13 byte.

Quando determinate i requisiti delle dimensioni del campo del record con l'ASSEMBLY del TMS9900, voi potete E dovete fare un uso molto più efficente dello spazio. Voi potete definire sia l'esatta posizione di inizio e fine dei campi, sia dove sono necessari i campi di lunghezza variabile, voi potete usare un carattere speciale per separare i campi, o designare un byte che precede ogni campo per contenere la lunghezza del campo stesso. Per i numeri, se si conosce che avranno un valore compreso in una gamma da 1 a 255, per esempio, allora il valore può essere memorizzato come un'espressione binaria di un solo byte.

Qualsiasi metodologia che lavora per voi può essere valida purchè voi siate tenaci. Usate la vostra creatività e la flessibilità dell'ASSEMBLY del TMS9900 per progettare file che rendono efficente l'utilizzazione dell'immagazzinamento e della memoria. Finchè i file di dati sono creati da voi per essere usati solo con altri

linguaggi ASSEMBLY del TMS9900, allora qualsiasi progetto sarà accettabile. Comunque, se voi intendete processare altri tipi di file, o rifornire i file da voi creati per altri tipi di programmi, allora voi dovete conoscere per quale specifica disposizione il file e record sono stati congegnati. Quando questa informazione non è disponibile, l'utilty DUMP dall'Aiuto alla Programmazione II può essere di grande aiuto. Questa utilty è decisamente raccomandata sia come un'utile strumento, sia come una chiave per capire come i record e file sono creati dal TI BASIC.

I file con record RELATIVE (ad accesso casuale) permettono l'accesso diretto a qualsiasi record sun file, senza che sia necessario leggere l'intero file, come nel caso di un file sequenziale. Questa caratteristica non solo può essere usata per accedere casualmente ad un singolo record, ma può anche essere usata per posizionare la testina di lettura/scrittura del disk drive, su un particolare record. Poi voi leggerete sequenzialmente i record seguenti. I file con record relative possono anche essere letti dal primo all'ultimo, o dall'ultimo al primo sequenzialmente, altrettanto bene. Tutti i file con record relative devono utilizzare record con lunghezza FIXED (fissa). I byte 4 e 5 del PAB (Lunghezza del record, e contatore dei caratteri) dovrebbero essere uguali.

La codificazione e l'istituzione del PAB e buffer per i file con record relative è identica a quella dell esempio precedente. Il progetto di una subroutine che esegua il DSR può essere esattamente simile a quella dell'esempio sequenziale. Il tipo di accesso che voi potete eseguire su un file relative dipende da come il file è stato aperto. INPUT vi permette solo di leggerlo. OUTPUT vi permette solo di scriverlo. UPDATE vi permetterà di leggere e scrivere. APPEND vi permetterà di aggiungere record alla fine del file precedentemente creato.

Le chiavi dell'accesso casuale (relative), è la manipolazione del numero dei record relative nei byte 5 e 6 del PAB. Se questi byte contengono tutti zeri alla vostra prima lettura, e fate letture successive senza alterare il numero del record relative, DSRLNK incrementerà questo valore per voi. Per conoscere quale record voi state per leggere o scrivere, ottenete il numero del record relative dai byte 6 e 7 del PAB prima di ogni DSRLNK. Ecco qui alcune istruzioni che fanno questo.

```
RELREC  BSS   2              Mette da parte spazio per i numeri


        LI    R0,PABADR+6    Indirizzo nel PAB del numero dei
*                            record relative
        LI    R1,RELREC
        LI    R2,2           Lunghezza da leggere, due byte
        BLWP  @VMBR         Ottiene il numero
```

Nella stessa maniera, se voi desiderate leggere un record specifico da un file relative, voi dovete mettere il numero del record relative nei byte 6 e 7 del PAB, prima che il DSRLNK sia eseguito. Le istruzioni sopra riportate faranno questo se voi sostituite VMBR con VMBW. Il risultato di provare a leggere un record relative che non esiste dovrebbe essere un errore di EOF (5) e DSRLNK chiuderà il file.

DSR CASSETTE

La routine DSR per le cassette è localizzata in GROM ed è una delle routine residenti GPL. Quando create file su cassette, devono essere osservate parecchie restrizioni. I file su cassette devono avere record di lunghezza fixed, e la loro lunghezza deve essere un multiplo di 64 (64, 128, o 192). I file su cassetta possono solo essere aperti come INPUT o OUTPUT. Con le cassette non esiste la rivelazione della fine del file (EOF). Voi dovete creare un record EOF come ultimo record su un file su cassetta, e codificare ogni programma che legge il file per rivelare qualunque EOF dei dati da voi creati. 73

Per accedere al DSR delle cassette o a qualsiasi altra routine GPL, bisogna includere una REF GPLLNK nel vostro programma, con l'E/A, o una direttiva EQU all'indirizzo >6018 con qualche etichetta valida di due caratteri, con l'assembler LINEA per LINEA. Usando la DSR delle cassette o qualsiasi altra routine GPL, la caratteristica partenza automatica (includendo l'indirizzo del punto di entrata con la direttiva END) non può essere usata.

La grande differenza con il trattamento dei file da cassetta, è la stessa DSR cassette. Ricordate quando codificate i dati del PAB per i vostri file su cassette di usare il valore corretto per il byte 8 del PAB (offset dello schermo). Per un programma ASSEMBLY autonomo , questo valore dovrebbe essere >00. Se il vostro programma in ASSEMBLY è chiamato dal BASIC, allora questo valore sarà >60. C'è qui un programma di esempio che usa un file su cassette. Questo programma scrive 10 record su un dispositivo a cassette. In verità questo programma non crea nessun dato valido per ogni record. Esso è soltanto un esempio di accesso alla cassetta.

```
01              REF   GPLLNK,VSBW,VMBW,KSCAN
02 STATUS       EQU   >837C
03 FAC          EQU   >834A
04 PABBUF       EQU   >1000
05 PAB          EQU   >F80
06 PDATA        DATA  >0002,>1000,>8000,>0000,>0003
07 DEV          TEXT  'CS1
08 RETURN       BSS   2
09 WR           BSS - >20
10 START        MOV   R11,@RETURN
11              LWPI  WR
12              LI    R0,PAB         Stabilisce il PAB
13              LI    R1,PDATA
14              LI    R2,14
15              BLWP  @VMBW
16              BL    @DSRCAS        Apre il file
17              LI    R1,>0300       Attiva la scrittura
18              LI    R0,PAB
```

```
19              BLWP  @VSBW
20              CLR   R6
21 CPUT         BL    @DSRCAS       Scrive un record
22              INC   R6            Aggiunge uno al contatore record
23              CI    R6,10         Controlla per 10 record scritti
24              JLE   CPUT
25              LI    R1,>0100      Attiva la chiusura del file
26              LI    R0,PAB
27              BLWP  @VSBW
28              BL    @DSRCAS       Chiude il file
29              JMP   EOJ           Vai alla fine del lavoro
30 DSRCAS       CLR   @STATUS       Pulisce il byte dello STATUS GPL
31              CLR   @>83D0        Pulisce l'indirizzo >83D0
32              LI    R3,3
33              MOV   R3,@>8354     Mette il valore 3 a >8354
34              MOV   @DEV,@FAC     Mette il nome dispositivo a FAC
35              MOVB  @DEV+2,@FAC+2
36              LI    R3,>0800
37              MOVB  R3,@>836D     Mette il valore 8 nel byte >836D
38              LI    R3,PAB+13
39              MOV   R3,@8356      Stabilisce il puntatatore al DSR
40              BLWP  @GPLLNK       Esegue la DSR della cassetta
41              DATA  >003D
42 *
43              RT                  Ritorna
44 EOJ          CLR   @STATUS       Pulisce lo STATUS
45              BLWP  @KSCAN        Attende che qualche tasto sia
   *                                 premuto
46              MOV   @STATUS,@STATUS
47              JEQ   EOJ
48              CLR   @STATUS
49              MOV   @RETURN,11
50              RT                  Ritorna
51              END
```


Il cuore di questo programma esempio è la routine della DSR cassetta dalla linea 30 alla linea 43. Il byte dello STATUS GPL e l'indirizzo >83D0 devono essere tutti a zero. Il nome del dispositivo ("CS1") deve essere messo all'indirizzo >834A, e la lunghezza del nome del dispositivo (3) deve essere a >8354, >8355. Poi il valore 8 deve essere messo nel byte all'indirizzo >836D per indicare una chiamata DSR. Il puntatore all'indirizzo del PAB deve essere messo all'indirizzo >8356, come negli altri esempi del DSR. Comunque, con il DSR delle cassette, il valore deve puntare al byte dopo il nome del dispositivo "CS1" (PAB+13). GPLLNK è richiesto con un'istruzione BLWP, e necessita del valore >3D passato ad esso per completare l'istruzione.

In questo programma esempio, l'apertura fa comparire il prompt "REWIND CASSETTE", La scrittura il prompt "PRESS CASSETTE RECORD", ed infine, la chiusura farà apparire il prompt "PRESS CASSETTE STOP".

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni sul trattamento dei file.

Sezione 16-2-2 pag. 251
Sezione 16-2-4 pag. 262
Dalla sezione 16-5 pag. 270, alla sezione 16-5-4 pag.271
Dalla sezione 18-1 pag. 291 alla sezione 18-3 pag. 303
Dalla sezione 24-12 pag. 443 alla sezione 24-12-4 pag. 444

NOTE SULL'ACCESSO ALLE GROM

1) L'accesso alle GROM/GRAM deve essere separato da almeno una istruzione per adattare le differenze dell'esecuzione hardware.

2) L'indirizzamento delle GROM è autoincrementante. Cioè, dopo ogni accesso all'indirizzo GROM, quest'ultimo viene automaticamente incrementato di 1 dal computer.

3) Gli indirizzi GROM sono scritti nel primo byte più significativo.

4) Quando l'indirizzo di lettura GROM/GRAM (GRMRA) è letto, l'indirizzo GROM viene distrutto e deve essere ristabilito se richiesto dal programma.

LEGGERE QUESTI TERMINI SUL GLOSSARIO

Device service routine = (routine dispositivo di servizio)
DSR
File
Field = (campi)
GPL
GROM
Mode of operation = (modo di operazione)
Peripheral Access Block = (blocco di accesso alle periferiche)
PAB

CAPITOLO DECIMO

CLASSIFICAZIONE E TRATTAMENTO DEGLI ARRAY

La sequenza dei dati in un prescritto ordine o "SORTing" =.(Classificazione), è un primo esempio di trattamento della tavola. Un tema ricorrente in questo testo è il limite del TI BASIC in velocità ed esecuzione rispetto all'ASSEMBLY del TMS9900. Forse nessun altro compito può illustrare meglio la sua velocità del SORTing. Viceversa il SORTing è uno dei migliori esempi sulle insufficenze del TI BASIC in termini di calcolo e velocità.

Il SORTing è un compito che coinvolge una gran parte del computer, perchè sono richieste un grande numero di azioni ripetitive. Ci sono molti sistemi di SORTing che cercano di limitare il numero di ripetizioni richieste per realizzare il SORT. Nessuno di questi sistemi richiede un trucco magico, ma fanno tutti assegnamento sul processore usato per elaborare rapidamente le azioni logiche richieste.

Come aumenta il numero di record, così aumenta il tempo richiesto per classificarli. Poichè la velocità e l'esecuzione sono critiche per l'esecuzione del SORT, la scelta logica di un linguaggio per scrivere un programma di SORT è l'ASSEMBLY.

Non è intenzione di questa sezione insegnarvi tutto sopra la teoria del SORT, ma piuttosto di illustrarvi come un semplice SORT può essere implementato usando il linguaggio ASSEMBLY del TMS9900, e le varie istruzioni, subroutine, e metodologie finora impiegate. Se voi volete diventare bravi nello i programmi di SORT, e imparare alcuni dei vari metodi utilizzabili per il SORTing, vi sono molti libri dedicati ad esso. Alcuni di questi sono:

Lorin Harold - SORTING AND SORT SYSTEMS
Knuth D.E. - THE ART OF COMPUTER PROGRAMMING
Flores Ivan - COMPUTER SORTING

Il SORT del programma di questa sezione non usa nessuna istruzione o routine che non sia già stata descritta in precedenza. Alcune di queste routine, i cui nomi voi riconoscerete sono state migliorate rispetto agli esempi precedenti, o sono usate in nuove e differenti maniere. Cose come titoli di programmi, prompt, grafici a colori, interazione con la tastiera, e trattamento dei file, sono tutti basati sugli esempi e le spiegazioni date nelle sezioni precedenti. Il presente SORT logico usato è piuttosto sommariamente confrontato ad alcune delle tecniche più avanzate che sono disponibili. Questo semplice sistema provvede ad un modello più facilmente comprensibile, e gira ancora abbastanza velocemente per merito della velocità del microprocessore TMS9900.

Il file dei dati da essere classificato è molto simile al nome del file del primo ed ultimo esempio della sezione precedente. Comunque per facilitare processi più veloci esso sarà un file relative con un record di lunghezza fixed di 80 byte. In computer più grandi, e con ben altre capacità di memoria, è possibile elaborare il maggior numero di file in qualsiasi formato, caricando i suoi dati in alcune aree di lavoro nella forma di una tavola, e manipolando i dati da li. Con il piccolo numero di record sul file esempio, questo metodo può essere usato con il computer casalingo. Il file nel formato relative, permette l'elaborazione di un file come se esso fosse già una tavola, e usa il dispositivo di memorizzazione (disco) per memorizzare la

tavola piuttosto che la CPU RAM.

Uno dei trucchi fondamentali di SORTing che sfugge a molti principianti, è che non si ha bisogno di fare il SORT di un'intero record di dati. Il solo dato che necessita di essere classificato, è quel dato che fa da chiave, per il campo o campi sui quali il file deve essere classificato. Questo provvede, naturalmente, che ciascuna di queste chiavi di SORT possano essere associate con il record al quale esso appartiene.

Questo è il punto dove il numero del record relative (il quale può essere una tavola ad indice) entra in gioco. La premessa base di questo programma è quella di creare una tavola interna, ogni ingresso della quale è comprensivo della chiave di SORT, e numero del record relative di ogni record. Questi dati sono poi classificati nella sequenza desiderata. La tavola viene poi processata sequenzialmente per ottenere ogni record al posto del numero del record relative, e un nuovo file della sequenza di classificazione viene scritto. La capacità di memoria di un dischetto semplice faccia/semplice densità è di circa 90 Kbyte. Questo totale è molto maggiore di quella disponibile nella CPU, così è impossibile caricare tutti questi byte dei record nella memoria in una sola volta. Con questo tipo di programma di SORT comunque, affrontare un file così grande è fattibile. In un programma ASSEMBLY un numero di record memorizzato sotto forma binaria occupa una word o 2 byte. Se il dischetto contiene 358 record di 250 byte ognuno, e i dati della chiave di SORT per ogni record è lunga 20 byte, allora il totale della memoria necessaria per la tavola di SORT dvrebbe essere: (Chiave della lunghezza del SORT + Lunghezza del record relative) * Numero dei record, o, (20 + 2) * 358 = 7.876 byte.

Alcuni sistemi di SORT permettono il SORT "in-place ?" del file originale. Questo programma crea invece un nuovo SORT del file, e preserva l'integrità del file originale, o Master. Un inconveniente del SORT "in-place" è che, se per qualche ragione, capita un errore nel processo, il file originale può essere rovinato o perduto, se non era stata fatta una copia di back-up del file. Questo sistema permette anche ai dati originali dei file di esistere in molti differenti file, ognuno nella sua propria sequenza. Questo può essere utile quando è necessario processare gli stessi dati da più di una chiave di SORT.

Ecco il programma completo del SORT

==

NOTA DEL TRADUTTORE:

Anche questo programma, essendo abbastanza lungo, viene allegato a parte per facilitare il controllo con le spiegazioni delle linee del programma stesso.

==

COMMENTI AL PROGRAMMA

Con quasi 400 linee questo programma ASSEMBLY TMS9900, può apparire complicato al principiante. Comunque, questo programma non è così complesso come il numero delle linee sembra indicare. In più, per eseguire il SORT logico, questo programma rappresenta un esempio di quanto finora è stato gii discusso in questo testo. Esso include un numero di opzioni e caratteristiche, oltre al SORT, che aumenta considerevolmente la sua lunghezza. Di fatto l'effettiva gamma di istruzioni del SORT vanno dalla linea 147 alla linea 251, cioè solo circa il 28% dell'intero programma. Il resto del programma è compreso dalle istruzioni, direttive, e subroutine che sono state precedentemente spiegate.

In ultimo, la chiave per comprendere qualsiasi programma che voi state guardando per la prima volta, è di esaminarlo linea per linea. E questo è vero sia per un programma lungo 5 o 5000 linee. Non siate intimiditi dalla lunghezza del programma. Le sole sezioni di questo programma da essere spiegate dettagliatamente, sono quelle che useranno una nuova logica (il SORT) e nuove o migliori applicazioni delle istruzioni e subroutine introdotte in precedenza.

Le linee da 1 fino a 59 usano le varie direttive assembler del TMS9900, per definire le costanti, gli indirizzi, dati e aree di lavoro che il programma richiederà. I numeri dei Data includono un nuovo descrittore del pattern per il cursore, caratteri grafici da visualizzare sullo schermo, dati per il PAB dell'input e output dei file, e i titoli, prompt, e messaggi da essere visualizzati. La linea 59 mette da parte 4096 byte (Hex >1000) per lo spazio della tavola del SORT. Questo è un valore arbitrario. Quando si progetta un programma di SORT o qualsiasi altra dimensione del programma interno del workspace, esso à necessario a permettere di prevedere alcuni o un massimo numero di record, e quindi, byte.

Le linee da 60 a 75 salvano l'indirizzo di ritorno, caricano il puntatore al registro workspace, fissano i colori di background e foreground del caratere del bordo, e visualizza i caratteri, caricando infine i dati del pattern del cursore nella tavola descrittrice del pattern in VDP RAM. Tutti questi passi ed istruzioni sono già state usate prima nei precedenti programmi esempio. I colori usati in questo programma sono il bianco sul blù, per il set di caratteri visualizzabili, e rosso scuro per il bordo dello schermo.

La linea 76 esegue un BL (Branch and Link) = (Diramati e allacciati) alla subroutine SCREEN. Questa subroutine caricherà il pattern del bordo dello schermo nella tavola immagine dello schermo, e scrive una linea di simboli "=" tra la parte alta e bassa dello schermo. Questa funzione viene codificata come subroutine per permettere allo schermo grafico di essere ristabilito o rifatto a qualsiasi punto del programma senza dover ricodificare le istruzioni.

Le linee da 79 a 90 visualizzano il titolo del programma e presentano il prompt per il nome del file/dispositivo per l'output del file da essere classificato. Mentre il nome File/dispositivo del file input difficilmente viene codificato dentro il programma, l'utente è libero di specificare l'output del file. Poichè il file output del SORT in questo programma esempio è un file relative, simile al file input, esso naturalmente deve essere su disco. Il nome del file /dispositivo viene letto dall'indirizzo 195 dello schermo, dentro il file output del byte 10 del PAB. La lunghezza di questi dati è ritornata in R7 da CURSOR. Questo valore della lunghezza viene mosso al file output nei byte 8 e 9 del PAB per completare i dati del file output.

Nel capitolo ottavo voi siete stati introdotti ad una routine fondamentale di input dalla tastiera, chiamata CURSOR, che provvedeva ad abilitare l'interrattività dalla tastiera. Questa routine è di nuovo inclusa in questo programma. Questa versione di CURSOR include alcune capacita aggiuntive che non esistevano nell'originale. Oltre che a usare un unico simbolo del pattern per il cursore, la nuova routine CURSOR lo fa anche lampeggiare. Due uscite addizionali sono permesse, all'utente. Se preme REDO = (rifare) la nuova routine CURSOR si dirama alla linea 76 (PROMPT) per reinserire nuovamente il dato richiesto. Se l'utente preme QUIT, la routine si dirama alla linea 268, EOJQ. Diramandosi a questo indirizzo, si salta la chiusura dell'input e output dei file (che non sono ancora stati aperti), e si termina il programma. Per ultimo, finchè la lunghezza della risposta specificata in R10 prima di BL a CURSOR è maggiore di zero, CURSOR non permetterà all'utente di premere semplicemente ENTER senza avere inserito alcuni dati.

Le linee da 91 a 107 chiedono all'utente di specificare quale campo del file deve essere classificato. I prompt definiti alle linee 34, 35, e 36 permettono all'utente di inserire "1" per il primo nome, o "2" per l'ultimo nome. La risposta viene confrontata con il codice ASCII di questi due numeri, per verificare se è stata fatta una scelta valida. Se viene inserito un qualsiasi valore oltre a quelli permessi, la sequenza del prompt viene ripetuta.

Questo ha lo stesso effetto di come se si premesse REDO. La sola altra opzione disponibile sarebbe QUIT. Poi le linee da 108 a 118 richiedono che l'utente specifichi l'ordine in cui i record devono essere classificati. La risposta "A" indica che i file devono esere classificati nella sequenza Ascendente, mentre la risposta "D" significa che la classificazione deve essere fatta in ordine Discendente. Le risposte vengono verificate perchè siano effettivamente "A" o "D". Giunti a questo punto il programma ha chiesto all'utente di indicare il nome del file/dispositivo del file output da classificare, quale campo classificare, e se il SORT deve essere in ordine Ascendente o Discendente.

Le linee da 119 a 123 chiedono all'utente di verificare le informazioni che sono state inserite, e rispondere premendo REDO o qualunque altro tasto. Se la risposta è REDO, allora il CURSOR si dirama all'inizio della sequenza prompt per permettere ai dati di essere reinseriti. Se qualsiasi altro tasto viene premuto , allora il programma continua. Le linee da 124 a 136 ottengono e salvano il campo e le sequenze delle opzioni, stabiliscono il PAB del file input in VDP RAM, aprono il file input, ed attivano il file input da leggere.

Le linee da 137 a 144 tolgono i prompt e le risposte, ripeteno lo schermo grafico, e visualizzano l'intestazione per il contatore dei record che saranno presi, e visualizzati durante il processo. Un altro contatore prenderà il numero dei record input (SORT in) e record output (SORT out). Questi due contatori dovranno essere sempre uguali, finchè il programma funziona bene, e non succedono errori.

La prima fase del processo di SORT implica la lettura del file che deve essere classificato, e la costruzione di una tavola per il campo del SORT e numero del record relative per ogni record. Poichè sono impiegate molte ripetizioni delle word (16 bit) nel caricare la tavola per eseguire il SORT effettivo, una subroutne chiamata MOVWRD è stata codificata proprio per questo scopo. Tutte le volte che lo stesso o similare gruppo di istruzioni necessita di essere eseguito più di una volta, è preferibile creare una subroutine comprendente il set di istruzioni, piuttosto che codificare le istruzioni stesse più e più volte ancora.

La routine MOVWRD usa i registri 6, 7, 8, e 9 per muovere le word di memoria tra le varie locazioni in CPU RAM. Il registro 7 deve contenere l'indirizzo di origine, il registro 8 l'indirizzo di destinazione, ed il registro 9, la lunghezza del movimento in byte (sempre un numero pari di word mosse), ed infine, il registro 6 è usato per i calcoli.

La linea 144 carica R8 con l'indirizzo di destinazione (area della tavola del SORT, SRTTAB). La linea 145 mette la lunghezza dei dati in R9. Le linee da 162 a 167 mettono l'indirizzo di origine in R7, dipendendo da quale campo è stato selezionato per essere classificato. La lunghezza del campo di SORT è fissata in 14 (>E) per ciascun campo. Questa è la massima lunghezza del primo campo del nome , come definito dal programma TI BASIC che lo ha creato. La lunghezza effettiva del campo dell'ultimo nome è variabile. La lunghezza del campo di SORT determina il numero dei byte che devono essere confrontati per rivelare quei record che sono fuori dalla sequenza.

Più grande è questo numero, e più tempo occorrerà per fare il SORT. Una scorciatoia può essere presa facendo un SORT logico solo per un dato numero di byte senza badare alla lunghezza del campo di SORT. Poichè è improbabile che qualsiasi record contenga un ultimo nome maggiore di 14 byte, è ancora più improbabile che i due ultimi nomi contengano gli stessi caratteri fino alla quindicesima posizione, questo metodo è ragionevolmente accurato, e risparmia del tempo. La lunghezza del valore potrebbe essere minore o maggiore, ma troppa differenza potrebbe essere inesatta, o richiedere più tempo. L'effettivo numero dei byte che voi scegliete per confrontare è soggettiva, ma è possibile barare un poco.

Per dimostrare come opera un SORT logico in questo programma, è necessario fare alcune ipotesi circa quali opzioni sono state selezionate, e provvedere ad alcuni dati per il processo del programma. Per il resto di questa spiegazione si presume che l'utente abbia bisogno di classificare il file di input sull'ultimo campo del nome, e abbia bisogno di classificare il file in ordine Ascendente. Ecco quì un file input di 10 record.

RECORD RELATIVE N°.	PRIMO NOME	ULTIMO NOME
0000	JIM	SMITH
0001	BOB	JONES
0002	MARY	QUEENS
0003	MARVIN	STONE
0004	LINDA	BLACKSMITH
0005	GEORGE	WASHINGTON
0006	THOMAS	JEFFERSON
0007	WILLIAM	MCKINLEY
0008	HOWARD	TAFT
0009	HOWARD	JOHNSON

Come voi potete vedere, questi record non sono in un ordine particolare. Le linee da 147 a 170 leggono il file e costruiscono la tavola del SORT. Ogni entrata della tavola è composta dai primi 14 byte dell'ultimo campo del nome, più 2 byte per il numero del record relative di ogni record, per un totale di una lunghezza di 16 byte. Quando viene raggiunta la fine della tavola, la linea 171 muove una word contenente >FFFF al prossimo indirizzo della tavola allo scopo di segnare la fine della tavola. La subroutine MOVWRD che viene usata per creare una tavola, illustra

l'uso di un modo di indirizzamento conosciuto come "WORKSPACE REGISTER INDIRECT AUTO-INCREMENT" = (Registro Workspace indiretto ad autoincremento). La linea 216 MOVEM *R7+,*R8+ specifica che il valore in ogni registro deve essere usato come indirizzo, e che dopo ogni movimento il valore nei registri sarà automaticamente incrementato. Questa utile caratteristica di indirizzamento, incrementa i registri implicati di 1 per le istruzioni byte, e di 2 per le word. Dopo che il file è stato letto, e la tavola di SORT creata, la tavola SORT dovrebbe apparire come sotto. Notate che gli effettivi valori nella tavola del SORT sono codici ASCII per le lettere, ed espressioni binarie per i numeri dei record relative. L'esempio è qui mostrato nel formato display.

INDIRIZZO	CONTENUTO DELLA TAVOLA	
SRTTAB+0	SMITH	00
SRTTAB+16	JONES	01
SRTTAB+32	QUEENS	02
SRTTAB+48	STONE	03
SRTTAB+64	BLACKSMITH	04
SRTTAB+80	WASHINGTON	05
SRTTAB+96	JEFFERSON	06
SRTTAB+112	MCKINLEY	07
SRTTAB+128	TAFT	08
SRTTAB+144	JOHNSON	09
SRTTAB+160	>FFFF	

La seconda fase di questo processo di classificazione (SORT), è di predisporre le entrate della tavola nell'ordine desiderato. Per fare questo, il programma parte con la prima entrata nella tavola, e confronta ogni byte dei dati della chiave di SORT con quello della prossima entrata della tavola. Se i dati della chiave di SORT delle due posizioni della tavola, confrontati fra loro, sono già nella giusta sequenza, o se sono uguali, allora non è richiesta nessuna azione, e il confronto si sposta alla prossima copia di entrate della tavola.

Per rendere facile mantenere la relazione delle entrate, da confrontare direttamente, la prima entrata della tavola sarà riferita come "A", e la prossima entrata come "B". Per l'esempio di sequenza Ascendente, tutte le volte che "A" è riconosciuto come maggiore di "B", le loro posizioni nella tavola verranno invertite.

Le linee da 180 a 185 calcolano l'indirizzo della copia di entrate della tavola da essere confrontate. L'indirizzo di "A" viene salvato in R0, mentre "B" è salvato in R1. Le linee 186, 187 esaminano l'indicatore di fine della tavola. Se la fine della tavola è stata raggiunta, allora questa fase del SORT è stata fatta, e la linea 187 dirige il programma all'indirizzo "DONE2". La prima volta per mezzo di questa logica, R3 è caricato con l'indirizzo SRTTAB, e R4 è caricato con l'indirizzo SRTTAB+16. R2 conterrà il valore dell'indirizzo dell'ultimo byte della parte della chiave SORT dell'entrata della tavola.

Se R2 è uguale a R3 alla linea 192, allora tutti i byte della chiave di SORT sono stati confrontati, ed è tempo di muovere alla prossima copia di entrate della tavola. Le linee 190 e 191 controllano l'opzione dell'ordine, confrontando ancora l'opzione con >44 (ASCII di "D"). Se è stato selezionato l'ordine discendente, allora viene eseguito il confronto con il ciclo CLOPD. Altrimenti, il default è l'ordine ascendente, il quale esegue il confronto con il ciclo CLOP. Ecco uno schema del primo confronto.

```
0194        CB  *R3+,*R4+
0195        JEQ CLOP
0196        JGT SWIT
```

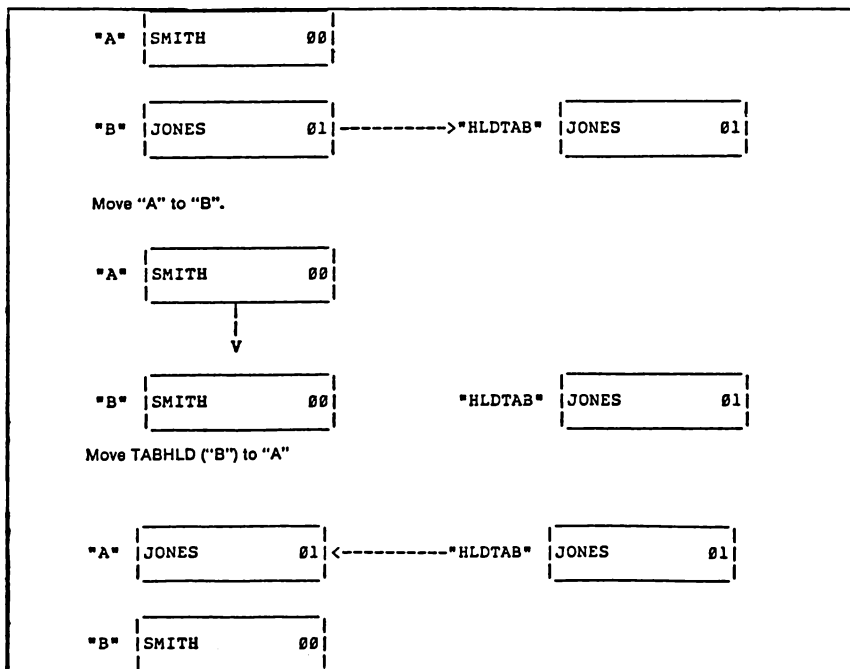
```
INDIRIZZO NEL              VALORE DELL'
REGISTRO 3                 INDIRIZZO
======================================
```



```
SRTTAB+0                   >53 O "S"

INDIRIZZO NEL              VALORE DELL'
REGISTRO 4                 INDIRIZZO
======================================

SRTTAB+16                  >4A O "J"
```

L'istruzione CB compara byte = (confronta i byte) è usata perchè il confronto deve essere fatto su un carattere dal carattere base, ed il carattere prende un byte. Il valore all'indirizzo in R3 ("S") è maggiore del valore all'indirizzo in R4 ("J"). Questo confronto fissa il bit "maggiore di" del registro di status. Questa condizione è controllata e l'azione risultante è diretta alla linea 196. Essa sarà necessaria per scambiare le posizioni delle due entrate delle tavole. Usando le nominate convenzioni "A", e "B", la tavola di entrata per SMITH è "A", e JONES sarà "B". Ecco tre passi per il processo di inversione.

Queste sono le istruzioni che eseguono l'inversione logica.

```
203 SWIT       MOV R1,R7      Carica R7 con l'indirizzo "B"
204            LI  R8,HLDTAB  Carica R8 con l'indirizzo HLDTAB
205            MOV @TABLEN,R9 Carica R9 con la lunghezza da muovere
206            BL  @MOVWRD    Muove "B" per occupare l'area
207            MOV R0,R7      Carica l'indirizzo di "A"
208            MOV R1,R8      Carica l'indirizzo di "B"
209            BL  @MOVWRD    Muove "A" a "B"
210            LI  R7,HLDTAB  Carica l'indirizzo da occupare
211            MOV R0,R8      Carica l'indirizzo di "A"
212            BL  @MOVWRD
213            JMP COMPAR     Controlla ancora la sequenza della
    *                         tavola
214 MOVWRD     MOV R9,R6      Carica la lunghezza da muovere
215            A   R7,R6      Calcola il massimo indirizzo
216 MOVEM      MOV *R7+,*R8+  Muove la word, e increm. l'indirizzo
217            C   R7,R6      Controlla il masssimo indirizzo
218            JNE MOVEM      Se non è massimo, muovi ancora
219            RT             Altrimenti ritorna
```

La linea 213 ritorna all'etichetta COMPAR dopo che l'inversione è stata fatta. Ogni volta che una coppia di entrate della tavola non sono trovate in ordine, e vengono invertite, il confronto logico ricomincia dall'inizio della tavola. Questo viene fatto più e più volte finchè tutte le entrate delle tavole sono nell'ordine corretto. Se tutte le entrate sono già nell'ordine corretto, non ci sarà mai una condizione che farà eseguire la inversione logica, e la fine della tavola è raggiunta. Il tipo di algoritmo del SORT applicato alla tavola, è conosciuto come "BUBBLE". Ogni voce nella tavola viene mossa alla posizione che gli appartiene. Voi potrete trovare molti programmi differenti che vengono tutti chiamati BUBBLE SORT. Le loro specifiche variano, ma il metodo generale è lo stesso. Quando questa seconda fase del programma di SORT è completata, la tavola del SORT assomiglierà a questa.

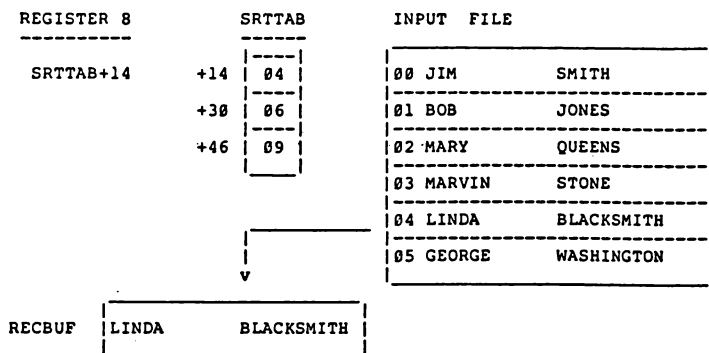
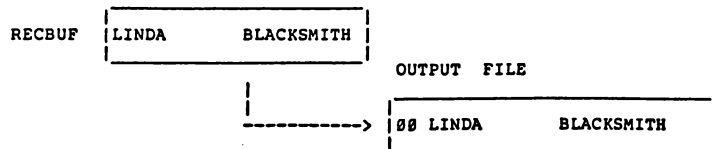
```
              |-----------------|
SRTTAB+0      |BLACKSMITH     04|
              |-----------------|
SRTTAB+16     |JEFFERSON      06|
              |-----------------|
SRTTAB+32     |JOHNSON        09|
              |-----------------|
SRTTAB+48     |JONES          01|
              |-----------------|
SRTTAB+64     |MCKINLEY       07|
              |-----------------|
SRTTAB+80     |QUEENS         02|
              |-----------------|
SRTTAB+96     |SMITH          00|
              |-----------------|
SRTTAB+112    |STONE          03|
              |-----------------|
SRTTAB+128    |TAFT           08|
              |-----------------|
SRTTAB+144    |WASHINGTON     05|
              |-----------------|
SRTTAB+160    |>FFFF            |
              |_________________|
```

La terza fase implica la lettura della tavola sequenzialmente, usando i numeri dei record relative per richiamare casualmente i record dal file input, e scriverli nel file output.

Le linee da 220 a 233 preparano la terza fase, stabilendo i PAB per i file input e output, aprendo i file input e output, e fissando l'OP-CODE per leggere il file input, e l'OP-CODE per scrivere il file output. Nella fase uno il file input era stato letto fino alla fine del file, ed era stato chiuso dalla routine DSR. Ora deve essere letto ancora, così i dati del PAB sono ristabiliti, ed il file riaperto.

La linea 234 carica l'indirizzo iniziale della tavola per il primo numero dell' entrata del record relativo in R8. Le linee da 235 a 238 controllano il FLAG = (indicatore) della fine della tavola, ogni volta per mezzo del ciclo GETRR. Le linee da 239 a 243 ottengono il numero del record relativo dalla tavola, lo mettono nel PAB byte 6 e 7 del file input, e leggono questo record dal file. I dati per il PAB per il file input, nominano l'area del VDP RAM, RECBUF come lo spazio buffer in cui ogni operazione di lettura mette un record del file input. I dati del PAB per il file output usa la stessa area buffer come il posto dove ogni operazione di scrittura si aspetta di trovare il record da scrivere. Nessun'altra manipolazione dei dati del record è richiesta.

Ecco lo schema del primo ciclo GETRR.

Ottiene (legge) il numero del record trovato all'indirizzo in R8 dal file input nel RECBUF.

Write an output file record from RECBUF.

Addiziona il valore a TABLEN (16, Hex >10) a R8, e ripete il ciclo. Il registro 8 ora contiene SRTTAB+30.

Questo processo continua finchè non viene raggiunto il segnalatore (Flag) di fine della tavola, ed a quel momento il programma logico si trasferisce a EOJ. Il messaggio "END OF JOB" = (fine del lavoro) viene allora visualizzato, ed il fle è chiuso. Il messagio "PRESS ANY KEY TO CONTINUE" = (premi un tasto per continuare), e un BL a CURSOR permette una pausa prima di finire effettivamente il programma. Con la possibilità di REDO di questa versione di CURSOR, l'utente a questo punto può scegliere di ripetere l'intero programma di SORT, o premere qualsiasi altro tasto per finire il processo.

Alla fine del programma il file input esiste nel suo stato originale, ed un nuovo file output del SORT è stato creato.

FILE INPUT			FILE OUTPUT		
RECORD Relativo	PRIMO Nome	SECONDO Nome	RECORD Relativo	PRIMO Nome	SECONDO Nome
0000	JIM	SMITH	0000	LINDA	BLACKSMITH
0001	BOB	JONES	0001	THOMAS	JEFFERSON
0002	MARY	QUEENS	0002	HOWARD	JOHNSON
0003	MARVIN	STONE	0003	BOB	JONES
0004	LINDA	BLACKSMITH	0004	WILLIAM	MCKINLEY
0005	GEORGE	WASHINGTON	0005	MARY	QUEENS
0006	THOMAS	JEFFERSON	0006	JIM	SMITH
0007	WILLIAM	MCKINLEY	0007	MARVIN	STONE
0008	HOWARD	TAFT	0008	HOWARD	TAFT
0009	HOWARD	JOHNSON	0009	GEORGE	WASHINGTON

La maniera in cui i requisiti input/output dei due file sono indirizzati, dimostra un modo in cui il DSR può venire applicato. Nella sezione del trattamento dei file, era stabilito che tutte le richieste DSR sono trattate alla stessa maniera, per qualsiasi dispositivo, eccetto le cassette, e la sola differenza erano i particolari dati del file (PAB) puntato, e le operazioni (op-code) richieste. Nella routine DSR di cui sopra, la sola differenza tra l'accesso al file input e quello output sono le linee 351, 352, e 353, che fissano il puntatore per il PAB (descrizione del file) usato. Gli op-code individuali per leggere, scrivere, e chiudere furono cambiati dentro il corpo del programma. Le istruzioni rimanenti lavorano per l'uno o l'altro file. Dovendo rivelare un errore, il puntatore all'indirizzo in R6 può essere usato per calcolare l'indirizzo del byte 1 del PAB (il codice di errore), ed il byte 10 del PAB (il descrittore del file "DSK1.FILE1") se il file output stava per essere scritto su un dispositivo a cassetta, questo metodo non lavorerà. Il DSR delle cassette deve essere accessibile da un differente set di istruzioni indicato nel capitolo del trattamento dei file.

La subroutine CURSOR come è codificata in questo programma di esempio, è una versione migliorata della prima introdotto. Questa versione opera in maniera molto simile all'originale. La differenza più notevole è il lampeggiare del cursore. Le

linee da 279 a 288 scrivono uno spazio (Hex >20) sullo schermo, e contano fino a 2.976 (Hex >0BA0) per quattro. Una volta che il ciclo CURL1 ha raggiunto questo valore, le linee da 289 a 297 visualizzano allora il simbolo del cursore. Il ciclo CURL2 conta fino a 2.976 per uno. Alcune indicazioni della velocità del linguaggio ASSEMBLY del TMS9900 viene evidenziata dal fatto che tutto questo conteggio venga fatto nello spazio di un lampeggio del cursore. Contando per quattro mentre lo spazio è visualizzato (), e contando per uno quando è visualizzato il cursore, significa che per l'80% del tempo viene visto il cursore, e per il 20% del tempo, lo spazio.

Voi potete anche alterare il ritmo del lampeggio, cambiando il valore a BLINK, o cambiando il valore di incremento per CURL1 o CURL2. L'utilità di scansione della tastiera viene eseguita dentro il primo ciclo, così la risposta alla pressione di un tasto non sarà troppo lenta.

Le linee da 298 a 301 controllano per i valori di REDO e QUIT, e dirigono di conseguenza il programma. Quando l'effettiva lunghezza del dato inserito è calcolata alla linea 332, il bit di uguaglianza del registro di status sarà fissato a uno, se il risultato in R7 è zero. Se un valore permesso di zero era messo in R10, prima che la routine logica di CURSOR fosse eseguita, allora le linee da 329 a 334 non saranno raggiunte. Comunque, se il CURSOR raggiunge l'etichetta ENTER, un valore maggiore di zero dovrà essere stato specificato. La linea 333 controlla per il valore in R7, e se esso è zero, l'intera subroutine di CURSOR viene ripetuta.

CAPITOLO UNDICESIMO

UNIRE L'ASSEMBLY CON IL BASIC

E possibile creare sottoprogrammi in linguaggio ASSEMBLY che possono essere chiamati dai vostri programmi in BASIC o EX/BASIC. Questo risulta un programma "ibrido" che offre il meglio di entrambi i linguaggi. I compiti del programma possono essere compiuti con adeguata efficenza in BASIC, e potrà essere poi codificato e "spulciato", (cioè corretto) facilmente con questo linguaggio.

I compiti che richiedono un miglioramento della velocità e esecuzione, o compiti che sono semplicemente impossibili da eseguire in BASIC, possono essere codificati in ASSEMBLY. Entrambi i moduli E/A o M/M provvedono ad aggiungere delle istruzioni supplementari per l'uso con programmi in BASIC. Queste istruzioni facilitano l'allacciamento delle routine ASSEMBLY con i programmi in BASIC. Allo stesso modo l'EX/BASIC offre altre istruzioni per facilitare queste funzioni che non sono normalmente disponibili in TI BASIC. Quando voi state girando un programma ASSEMBLY con l'E/A o la M/M, la memoria del computer non è disposta esattamente allo stesso modo come quando voi operate in BASIC o EX/BASIC. Allo scopo di far funzionare correttamente i vostri programmi ASSEMBLY, in ambiente BASIC, dovrete conoscere alcune differenze specifiche tra i due linguaggi.

Primo, voi dovrete conoscere in quale ambiente BASIC il vostro programma opererà. Di primaria considerazione per il programmatore ASSEMBLY sono le differenze trovate in VDP RAM. Il programma BASIC e tutti i suoi dati, routine, e workspace occupano una buona parte di questa area. Il vostro programma ASSEMBLY non deve cambiare i valori di qualsiasi indirizzo usato dal BASIC. Certi valori delle tavole in VDP RAM, come la Tavola del colore, sono locate a differenti indirizzi in BASIC.

Inoltre, il vostro programma ASSEMBLY deve permettere un offset (differenza) dello schermo quando gira con il BASIC. Il valore di questo offset è sempre di >60 (Dec. 96). Qualsiasi carattere che desiderate visualizzare con il vostro programma ASSEMBLY deve avere il valore del codice del carattere aumentato di >60 Normalmente, per visualizzare una "A" sullo schermo, il codice del carattere è >41. In ambiente BASIC, comunque, questo valore deve essere aggiustato

```
LI   R0,293    Carica R0 con l'indirizzo dello schermo in VDP RAM
LI   R1,>4100  Carica R1 con il codice di "A"
AI   R1,>6000  Aggiusta per il BASIC
BLWP @VSBW    Visualizza la lettera "A"
```

Per le costanti di stringa come quelle create con la direttiva TEXT ogni byte della stringa deve essere aggiustato aggiungendo >60 prima di scriverla all'indirizzo della Tavola Immagine dello Schermo (TIS).

Ecco un semplice programma dal capitolo ottavo. Questa versione incorpora cambiamenti necessari perchè questo programma ASSEMBLY sia chiamato dal BASIC.

```
        DEF  GO
        REF  VWTR,VSBW,VMBW,KSCAN
WR      BSS  >20
STATUS  EQU  >837C
KEYVAL  EQU  >8375
DTEN    DATA >A

==============================================================

* Tutti i codici del carattere spazio (>20) sono stati incrementati
* di >60

BORDER  DATA >FFFF,>8080,>8080,>8080
        DATA >8080,>8080,>8080,>8080
        DATA >8080,>8080,>8080,>8080
        DATA >8080,>8080,>8080,>FFFF

==============================================================

MSG1    TEXT '** PRESS ANY KEY    **'
MSG2    TEXT '* KEYSTROKE VALUE IS *'
MSG3    TEXT '* PRESS REDO/ESCAPE  *'

==============================================================

* Offset della costante per il BASIC

OFFST   BYTE >60

==============================================================

REDOV   BYTE >06
ESCV    BYTE >0F
        EVEN      Forza una word pari nel contatore di locazione
SAV11   BSS  2
GO      MOV  R11,@SAV11
        LWPI WR
        LI   R0,>0755
        BLWP @VWTR

==============================================================

* Gli indirizzi usati in queste istruzioni per accedere alla tavola
* del colore in VDP RAM sono stati aggiustati per il BASIC.

        LI   R0,799
        LI   R1,>5500
        BLWP @VSBW
        LI   R0,780
```

```
        LI   R1,>1F00
CLOOP   BLWP @VSBW
        CI   R0.798
        JEQ  BPUT
        INC  R0
        JMP  CLOOP

======================================================================

BPUT    LI   R0,0
        LI   R1,BORDER
        LI   R2,32
BLOOP   BLWP @VMBW
        CI   R0,736
        JEQ  EXIT
        AI   R0,32
        JMP  BLOOP
EXIT    LI   R0,261    Indirizzo VDP RAM per il primo messaggio
        LI   R2,MSG1   Indirizzo del messaggio in CPU RAM
        LI   R3,22     Lunghezza del messaggio
        BL   @PBASIC  Stampa il messaggio
SCAN1   CLR  @STATUS
        BLWP @KSCAN
        MOVB @STATUS,@STATUS
        JEQ  SCAN1
        LI   R0,325    Indirizzo VDP RAM per il secondo indirizzo
        LI   R2,MSG2   Indirizzo del messaggio in CPU RAM
        LI   R3,22     lunghezza del messaggio
        BL   @PBASIC  Stampa il messaggio
        LI   R0,395
        MOVB @KEYVAL,R1

======================================================================

* Il valore del tasto premuto deve essere aggiustato di >60 prima che
* sia visualizzato

        AB   @OFFST,R1

======================================================================

        BLWP @VSBW
        CLR  R4
        MOVB @KEYVAL,R4
        SRL  R4,8
        LI   R3,404
        LI   R0,406
        BL   @FIGUR
        LI   R0,485    Indirizzo VDP RAM per il terzo messaggio
        LI   R2,MSG3   Indirizzo del messaggio in CPU RAM
        LI   R3,22     Lunghezza del messaggio
        BL   @PBASIC  Stampa il messaggio
```

```
SCAN2    CLR  @STATUS
         BLWP @KSCAN
         MOVB @STATUS,@STATUS
         JEQ  SCAN2
         CB   @KEYVAL,@ESCV
         JNE  SCAN2
         B    @BPUT
FIGUR    MOV  R4,R5
         CLR  R4
         DIV  @DTEN,R4
         AI   R5,>0030
         SLA  R5,8
         MOV  R5,R1

=====================================================================

* Il numero ASCII deve essere aggiustato di >60 prima di venire
* visualizzato

         AB   @OFFST,R1

=====================================================================

         BLWP @VSBW
         DEC  R0
         C    R0,R3
         JHE  FIGUR
         RT
ESCAP    CLR  @STATUS
         MOV  @SAV11,R11
         RT

=====================================================================

* La routine "PBASIC" addiziona l'offset per il BASIC ad ogni byte
* dei dati da visualizzare, e scrive un byte alla volta sullo schermo

=====================================================================

PBASIC   MOVB *R2+,R1     Muove un byte del messaggio, in R1
         AB   @OFFST,R1 Aggiusta per il BASIC
         BLWP @VSBW     Scrive un byte
         DEC  R3          Decrementa il contatore dei caratteri
         JNE  PBASIC      Se non è zero, ricomincia ancora
         RT

=====================================================================

         END
```

Ora per chiamare questa routine da un programma BASIC, voi dovreste

scrivere un programma in questo formato.

```
10 REM PROGRAMMA IN BASIC PER CHIAMARE
20 REM IL PROGRAMMA DEL CAPITOLO OTTAVO
30 REM IN LINGUAGGIO ASSEMBLY
40 CALL INIT
50 CALL LOAD("DSK1.OBJECT")
60 CALL LINK("GO")
70 CALL SCREEN(4)
80 FOR LOOP=780 TO 799
90 CALL POKEV(LOOP,19)
100 NEXT LOOP
110 GOTO 60
```

Per gli utenti della M/M che hanno già assemblato il programma, memorizzato e addizionato nel modulo il nome del programma ed il punto di entrata per la tavola REF/DEF, l'istruzione LOAD della linea 50 non è necessaria. CALL INIT inizializza la memoria, pulisce i programmi e dati caricati in precedenza e controlla per vedere se l'espansione di memoria è collegata, e se è così, inizializza i valori necessari per allacciarla alla memoria della consolle per usare il linguaggio ASSEMBLY. CALL INIT deve apparire nel vostro programma PRIMA del primo CALL LOAD.

CALL LOAD carica il codice oggetto dal file name del dispositivo specificato. Esso può anche essere usato per mettere (o "poke") i valori nella CPU RAM nel formato CALL LOAD (Indirizzo,Valore1,Valore2,Valore3...). CALL LINK si dirama all'indirizzo del punto di entrata del programma in linguaggio ASSEMBLY. Il controllo a questo punto viene passato al sottoprogramma. L'esecuzione delle rimanenti istruzioni in BASIC non continuano finchè il programma ASSEMBLY non ha completato il suo incarico e ritorna.

Il programma ASSEMBLY di questo esempio, compone lo schermo e il colore dei caratteri come bianchi su fondo blu. Ritornando dal sottoprogramma , il programma BASIC rimette lo schermo ed il colore dei caratteri come neri su fondo verde.

In più usando CALL LOAD per mettere i valori nella CPU RAM, in TI BASIC sono disponibili altre utili istruzioni con l'E/A e M/M.
CALL PEEK (Indirizzo,Variabile1,Variabile2,Variabile3,...) vi permette di recuperare uno o più valori da un'indirizzo in CPU RAM. Similarmente CALL PEEKV (Indirizzo,Valore1,Valore2,Valore3,...) recupera un valore dal VDP RAM. CALL POKEV (Indirizzo,Valore1,Valore2,Valore3,...) mette uno o più valori in un'indirizzo nella VDP RAM.

Il formato per ciascuna di queste istruzioni è lo stesso. I valori sono memorizzati nel formato decimale. Indirizzi maggiori di 32.767 sono espressi come negativi in notazione di complemento a due. Ciascuna variabile o valore indica un dato di un byte. Quando viene usato più di un valore o variabile, il primo è assegnato al byte all'indirizzo specificato, e ciascun valore o variabile successiva viene assegnato dal prossimo byte al prossimo indirizzo.

L'istruzione POKEV rimette i valori dei colori di primo piano e di sfondo della tavola dei colori del programma esempio di cui sopra, come nero su verde. Ogni byte

nella tavola del colore è fissato a >13 o dec. 19. Con la vostra conoscenza delle tavole e indirizzi, usando POKE e PEEK nei vostri programmi BASIC potrete eseguire certi incarichi più velocemente che con le normali istruzioni BASIC, e fare anche cose normalmente impossibili col BASIC. Ricordate che l'offset dello schermo, di >60 (dec. 96) riguarda anche queste istruzioni. Per esempio, CALL POKE (2,161) visualizza una "A" alla riga 1 e colonna 3. Sebbene il normale codice ASCII di "A" sia 65, aggiungendo ad esso l'offset di 96, si ha appunto 161 (96+65).

I byte >837D, >837E, >837F in CPU RAM sono parte di un'area conosciuta come CPU RAM PAD. Questi indirizzi creano il buffer caratteri in VDP. Mettendo il codice del valore del carattere a 837D, il carattere sarà visualizzato sullo schermo alla riga e colonna specificata a >837E, e >837F rispettivamente. Poichè l'indirizzo >837D (dec. 33661) è maggiore di 32.767, l'indirizzo deve essere dato nella notazione in complemento a due. Inoltre, questi indirizzi sono in CPU RAM e possono essere accessibili con CALL LOAD. In questa maniera, alcuni accessi al VDP potranno essere ottenuti senza usare POKEV, il quale non è disponibile in EX-BASIC. CALL LOAD (-31875,161,1,3) visualizza una "A" alla riga 1 e colonna 3, proprio come ha fatto POKEV nell'esempio più sopra. Il primo valore (161) è messo all'indirizzo >837D (dec. -31875), il secondo valore (1) viene messo all'indirizzo >837E (dec. -31874), ed il terzo valore (3) è messo all'indirizzo >837F (dec. -31873).

L'AMBIENTE EXTENDED BASIC

Esistono delle differenze su come un programma ASSEMBLY può essere usato con l'EX-BASIC in confronto al BASIC. Particolarmente, l'uso del VDP RAM varia sia rispetto al BASIC, sia rispetto al modo ASSEMBLY. Inoltre, in CPU RAM, l'area occupata dal'unità di espansione della memoria, può essere usata dall'EX-BASIC. Gli indirizzi da >2000 a >3FFF, e da >A000 a >FFE0 sono usati dal caricatore (LOADER) dell'EX-BASIC. I programmi in linguaggio ASSEMBLY possono essere solo più lunghi di 8Kbyte dell'EX-BASIC.

UTILIZZAZIONE VDP RAM
IN MODO EX-BASIC

INDIRIZZI

HEX	DECIMALI	
>0000	0	Tavola Immagine
>02FF	767	Schermo
>0300	768	Lista Attributi
>03FF	1023	Sprite
>0400	1024	Descrizione Pattern e Tavola Descrizione
>077F	1919	SPRITE

```
==================================
 >0780          1920  Tavola
                      Movimento
 >07FF          2047  Sprite
==================================
 >0800          2048  Tavola
 >081F          2079  Colore
==================================
 >0820          2080  Interprete
                      Programmi
                      Ex-Basic,
                      Area lavoro,
                      Tavola dati,
 >3FFF         16383  etc. etc.
==================================
```

Il caricatore dell'EX/BASIC non riconosce i riferimenti esterni (REF). Per accedere alle utilità, come VSBW, gli indirizzi delle utilità devono essere EQUagliate con i loro nomi. Queste utilità sono locate a indirizzi diversi di quelli usati in TI BASIC o ASSEMBLY, e non tutte le utilità sono supportate. Per esempio, DSRLNK non viene supportata sotto l'EX-BASIC. Quando il vostro programma ASSEMBLY sta girando dall'EX-BASIC, o quando esso gira automaticamente, avendo incluso il punto di entrata dell'etichetta con la direttiva END, esso partirà nel workspace GPL. Il workspace GPL parte a >83E0. Voi non dovete usare questa area come vostro workspace. Piuttosto, voi dovreste stabilire il vostro workspace, definendolo con l'istruzione LWPI all'inizio del vostro programma. Questo è stato fatto fin'ora in tutti i programmi esempio, e generalmente è una buona pratica. Allo scopo di far ritornare in maniera appropriata il vostro programma al programma che lo ha chiamato, voi dovreste il registro puntatore al workspace così che esso punti al workspace GPL prima di ritornare.

Le seguenti istruzioni illustrano come questo viene fatto.

```
GPLWS    EQU    >83E0        Indirizzo iniziale del Workspace GPL
SAV11    BSS    2            Salva l'area per l'indirizzo di ritorno
MYWS     BSS    >20          Mette da parte 32 byte per il mio WS
*
*
START    MOV    R11,@SAV11   Salva l'indirizzo di ritorno
         LWPI   MYWS         Stabilisce il mio Workspace
*
*
END      LWPI   GPLWS        Ristabilisce il Workspace GPL
         MOV    @SAV11,R11   Restituisce l'indirizzo di ritorno
         CLR    STATUS       Pulisce il byte dello STATUS GPL
         RT                  Ritorna
```

Ecco un'altra maniera di finire il programma che lavora in qualsiasi ambiente, non importa se il vostro programma ASSEMBLY è chiamato da BASIC o EX-BASIC. Per questo metodo, i contenuti di R11 non necessitano di essere salvati. Le seguenti istruzioni illustrano questa tecnica.

```
GPLWS    EQU   >83E0
*
*
END      LWPI  GPLWS
         CLR   @STATUS
         B     @>0070
```

Ancora una volta, riecco il programma esempio del capitolo ottavo. Questa volta, esso è stato scritto con i cambiamenti necessari per chiamarlo dall'EX-BASIC.

```
        DEF   GO

================================================================
* Il caricatore (LOADER) dell'EX-BASIC non riconosce i riferimenti
* esterni (REF). Le utilitfà VWTR, VSBW, VMBW, E KSCAN sono
* accessibili includendo le direttive EQU. Gli indirizzi delle
* utilità sono differenti in ambiente EX-BASIC.
================================================================

VWTR    EQU   >2030
VSBW    EQU   >2020
VMBW    EQU   >2024
KSCAN   EQU   >201C

================================================================
* "GPLWS" Indirizzo del workspace GPL
================================================================

GPLWS   EQU   >83E0
WR      BSS   >20
STATUS  EQU   >837C
KEYVAL  EQU   >8375
DTEN    DATA  >A
BORDER  DATA  >FFFF,>8080,>8080,>8080
        DATA  >8080,>8080,>8080,>8080
        DATA  >8080,>8080,>8080,>8080
        DATA  >8080,>8080,>8080,>FFFF
MSG1    TEXT  '** PRESS ANY KEY     **'
MSG2    TEXT  '* KEYSTROKE VALUE IS  *'
MSG3    TEXT  '* PRESS REDO/ESCAPE   *'
OFFST   BYTE  >60
REDOV   BYTE  >06
ESCV    BYTE  >0F
        EVEN
GO      LWPI  WR
        LI    R0,0755
        BLWP  @VWTR

================================================================
* Gli indirizzi usati da queste istruzioni per accedere alla tavola
* del colore, che è stata cambiata.
================================================================
```

```
        LI      R0,>081F
        LI      R1,>5500
        BLWP    @VSBW
        LI      R0,0800
        LI      R1,>1F00
CLOOP   BLWP    @VSBW
        CI      R0,>081E
        JEQ     BPUT
        INC     R0
        JMP     CLOOP
BPUT    LI      R0,0
        LI      R1,BORDER
        LI      R2,32
BLOOP   BLWP    @VMBW
        CI      R0,736
        JEQ     EXIT
        AI      R0,32
        JMP     BLOOP
EXIT    LI      R0,261
        LI      R2,MSG1     Indirizzo in CPU RAM del messaggio
        LI      R3,22       Lunghezza del messaggio
        BL      @PBASIC   Stampa il messaggio
SCAN1   CLR     @STATUS
        BLWP    @KSCAN
        MOVB    @STATUS,@STATUS
        JEQ     SCAN1
        LI      R0,325      Indirizzo VDP RAM per il secondo messaggio
        LI      R2,MSG2     Indirizzo in CPU RAM del messaggio
        LI      R3,22       Lunghezza del messaggio
        BL      @PBASIC   Stampa il messaggio
        LI      R0,395
        MOVB    @KEYVAL,R1
        AB      @OFFST,R1
        BLWP    @VSBW
        CLR     R4
        MOVB    @KEYVAL,R4
        SRL     R4,8
        LI      R3,404
        LI      R0,406
        BL      @FIGUR
        LI      R0,485      Indirizzo VDP RAM per il terzo messaggio
        LI      R2,MSG3     Indirizzo in CPU RAM del messaggio
        LI      R3,22       Lunghezza del messaggio
        BL      @PBASIC   Stampa il messaggio
SCAN2   CLR     @STATUS
        BLWP    @KSCAN
        MOVB    @STATUS,@STATUS
        JEQ     SCAN2
        CB      @KEYVAL,@REDOV
        JNE     SCAN2
        B       @BPUT
FIGUR   MOV     R4,R5
```

```
        CLR   R4
        DIV   @DTEN,R4
        AI    R5,>0030
        SLA   R5,8
        MOV   R5,R1
        AB    @OFFST,R1
        BLWP  @VSBW
        DEC   R0
        C     R0,R3
        JHE   FIGUR
        RT

=====================================================================
* "ESCAP" Rimette a zero (Resetta) il puntatore al registro per il
* workspace GPL, pulisce il byte di STATUS, e si dirama all'indirizzo
* >0070
=====================================================================

ESCAP   LWPI  GPLWS
        CLR   @STATUS
        B     >0070
PBASIC  MOVB  *R2+,R1      Muove un byte del messaggio in R1
        AB    @OFFST,R1   Aggiusta per il BASIC
        BLWP  @VSBW       Scrive un byte
        DEC   R8           Decrementa il contatore caratteri
        JNE   PBASIC       Se non è zero, ripeti
        RT                 ...altrimenti ritorna
        END


10 REM * PROGRAMMA IN EX-BASIC PER CHIAMARE *
20 REM * IL PROGRAMMA ASSEMBLY DEL          *
30 REM * CAPITOLO OTTAVO                    *
40 CALL INIT
50 CALL LOAD("DSK1.OBJECT")
60 CALL LINK("GO")
70 CALL SCREEN(4)
80 FOR LOOP=0 TO 16
90 CALL COLOR(LOOP,2,4)
100 NEXT LOOP
110 GOTO 60
```

In più, avendo programmi in BASIC e EX-BASIC chiamati dal programma ASSEMBLY, voi potete passare dati numerici e stringhe dall'uno all'altro. Questo può essere fatto con PEEK e POKE, o può essere usata l'istruzione CALL LINK. L'indirizzo del punto di entrata in CALL LINK può essere seguito da un massimo di 16 variabili, che sono disponibili nel linguaggio ASSEMBLY per agire su di esso.

Il seguente programma esempio simula l'istruzione DISPLAY AT dell'EX-BASIC. Esso è scritto per essere girato dal TI-BASIC, con il modulo della M/M o dell'E/A.

Il formato di un programma in TI-BASIC che userà questo sottoprogramma varierà in dipendenza dal tipo di modulo che viene usato, e da quale configurazione Hardware userete con il vostro computer.

Per gli utenti della M/M, chi ha già assemblato il programma, memorizzato nel modulo, e aggiunto il nome del programma ed il punto di entrata alla tavola REF/DEF, dovrebbero usare questo formato:

```
CALL LINK("DEF LABEL",RIGA,COL,STRINGA)
```

Per esempio

```
110 CALL LINK("GO",12,6,S$)
```

Gli utenti dell'E/A dovrebbero usare un formato simile a questo programma in TI-BASIC.

```
10 REM * PROGRAMMA IN BASIC PER CHIAMARE *
20 REM * IL PROGRAMMA ASSEMBLY           *
30 REM * "DISPLAY AT"                    *
40 CALL INIT
50 CALL LOAD("DSK1.BSCSUP")
60 CALL LOAD("DSK1.OBJECT")
70 INPUT "STRINGA?":S$
80 INPUT "RIGA?":R
90 INPUT "COL?":C
100 CALL LINK("GO",R,C,S$)
110 FOR DELAY=1 TO 500
120 NEXT DELAY
130 GOTO 70
```

Le routine speciali usate in questo programma ASSEMBLY, conosciute come "Supporto al BASIC" (NUMREF, STRREF, ERR) sono incluse nel dischetto "A" dell'E/A. Queste devono essere caricate nel computer con un'istruzione BASIC come quella qui usata, CALL LOAD("DSK1.BSCSUP"), allo scopo di determinare i nomi simbolici di queste routine per il programma ASSEMBLY. Queste routine esistono già nel modulo della M/M, così questa istruzine non è richiesta. Poi il file oggetto che è stato creato assemblando il programma viene caricato. Voi dovreste specificare il nome del dispositivo che si adatta alla vostra situazione.

NOTA DEL TRADUTTORE:

Anche questo programma viene listato a parte, per permettere un migliore controllo con le spiegazioni del programma.

Spiegazioni del programma "DISLPAY AT".

Le linee 8 e 9 puliscono R0 e caricano R1 con il valore di 1. Questi registri sono usati dalle routine NUMREF e STRREF come indicatori. Il valore di R0 dice alla routine che tipo di valore essa ritroverà. Il valore di R1 dice alle routine quale variabile deve essere ritrovata. Un "1" indica il primo valore passato dal programma BASIC all'istruzione CALL LINK. Il primo valore, in quest'esempio, è il numero della riga. La linea 10 esegue una diramazione ed un'allacciamento (BL) alla routine "GETNUM" alla linea 32.

NUMREF ottiene il valore (che è nel formato in virgola mobile) dal valore di STAK del TI-BASIC, e lo mette all'indirizzo >834A. Questo indirizzo è il "FLOATING POINT ACCUMULATOR" FAC = (Accumulatore in virgola mobile). Poi, la routine XMLLNK prende il numero e lo converte in intero (in effetti, un'espressione binaria su cui può agire un programma ASSEMBLY). Il risultato di questa conversione è tuttora all'indirizzo del FAC (>834A).

Bisogna poi verificare che questo numero di riga sia valido (da 1 a 24). La linea 11 esegue un BL alla routine "CHKLMR" della linea 39. Prima il valore viene confrontato al valore di LIM+2, che è 24. Poi il valore viene confrontato col valore a LIM che è 1. Se il valore della riga è maggiore di 24, o minore di 1, allora viene fatto un salto all'etichetta ERROR della linea 44. La routine di errore causa il messaggio "BAD VALUE" = (Valore errato), che sarà visualizzato, e ritornerà al programma BASIC.

La linea 12 del programma ASSEMBLY viene raggiunta solo quando il valore della riga è stato verificato come valido. La linea 12 muove il valore della riga in R4. Il valore necessario per calcolare l'indirizzo dello schermo in relazione al linguaggio ASSEMBLY per questa riga è di uno in meno dell'effettivo valore recuperato. La linea 13 aggiusta per questa condizione DECrementando R4. L'indirzzo dello schermo per questa riga è uguale al valore ora in R4, moltiplicato per 32. La linea 14 compie questa operazione spostando i bit di R4 a sinistra di 5 posizioni. Ogni volta che i bit di un registro sono spostati a sinistra, l'effetto sul valore del registro è uguale a: VALORE * 2 N, dove N è il numero delle posizioni spostate a sinistra. Poichè R4 viene spostato a sinistra di 5 posizioni, e 2 5 è uguale a 32, si ha lo stesso effetto come moltiplicando per 32. Usare l'istruzione di spostamento a sinistra in questa maniera è un'abile sistema per eseguire le moltiplicazioni, se il valore che voi desiderate moltiplicare è una potenza di 2. il valore ora in R4 è l'indirizzo dello schermo della riga selezionata. La linea 15 salva questo valore in R7.

La linea 16 aggiunge 1 al valore in R1 prima che le linee 17 e 18 ritrovino e verifichino il numero della colonna. Addizionando 1 a R1 si dice alle routine NUMREF E STRREF di operare sul prossimo valore passato da CALL LINK. La linea 19 viene raggiunta solo una volta che il numero della colonna è stato recuperato e verificato. Esso adiziona il valore della colonna all'indirizzo dela riga in R4. La linea 20 addiziona 1 a questo valore per farlo corrispondere al modo in cui DISPLAY AT tratta

gli indirizzi di riga e colonna.

Normalmente, quando l'utente seleziona la riga 1 e colonna 1, questo trasferisce all'indirizzo 0 della Tavola Immagine dello Schermo, (TIS) la prima posizine sullo schermo. Comunque, DISPLAY AT riconosce solo le colonne da 3 a 30 di ogni riga, per una lunghezza di 28 byte. Cosi, se viene selezionata la riga 1 e la colonna 1, la visualizzazione effettiva comincia dalla riga 1 e colonna 3. Le colonne 1, 2, 31, e 32 di ogni riga contengono caratteri di riempimento, e non sono usate per visualizzare. Il registro quattro ora contiene l'indirizzo della TIS in VDP RAM, che corrisponde al numero della riga e colonna selezionata.

Alla linea 21, R1 è ancora incrementato per indicare a STRREF che esso deve operare sul prossimo valore passato da CALL LINK. L'utilità STRREF recupera la stringa BASIC, e la converte in una stringa in linguaggio ASSEMBLY. La linea 22 carica R2 con l'indirizzo di dove la stringa deve finire (SBUF).

Il primo byte del buffer di stringa deve essere un valore che fissa il massimo numero di byte da accettare. Quando la stringa è ritornata da STRREF essa partirà dal byte SBUF+1, ed il primo byte (SBUF+0) sarà cambiato per riflettere l'effettiva lunghezza della stringa. Il massimo numero di byte accettati è di 255, o >FF. La linea 23 usa l'istruzione SETO, la quale fissa a uno tutti i bit nell'indirizzo nominato. Questo è l'effetto opposto dell'istruzione CLR che fissa a zero tutti i bit. Se tutti i bit di una Word in memoria sono fissati a uno, il valore Hex di questa Word è >FFFF. La linea 24 esegue il BLWP a STRREF, che effettivamente ottiene la stringa. La linea 26 muove l'indirizzo del buffer stringa a R3 poichè R2 sarà usato più tardi. La linea 27 muove la lunghezza della stringa trovata nel primo byte del buffer stringa a R5, il quale era stato pulito alla linea 25. La linea 27 usa la caratteristica di indirizzamento con autoincremento, lasciando che l'indirizzo in R3 punti a SBUF+1 dopo che è stato mosso. Se la lunghezza della stringa è zero quando viene mossa in R5, il bit di uguaglianza verrà messo a uno nel registro di STATUS. Questa condizione viene controllata alla linea 28, che dirige la logica al programma di chiamata, poichè una lunghezza zero indica che nessuna stringa è stata passata. La linea 29 usa l'istruzione SWAP BYTE (SWPB) = (Scambia i byte), per cambiare il valore in R5 dal byte sinistro al byte destro. Se il valore mosso in R5 alla linea 27 era >FF, allora R5 conterrà >00FF. Dopo l'istruzione SWPB, R5 contiene >FF00, o 255 Dec. La linea 30 esegue un BL alla routine "PRINT" che visualizzerà la stringa alla locazione desiderata sullo schermo.

La linea 47 carica R6 con il valore di offset dello schermo che deve essere aggiunto a qualsiasi codice dei caratteri ASCII prima che essi possano essere visualizzati da un programma ASSEMBLY che è stato chiamato dal BASIC. La linea 48 addiziona 30 al valore in R7. Il valore in R7 era l'indirizzo dello schermo della riga selezionata. R7 ora contiene il valore dell'indirizzo dello schermo della 31ma. colonna della riga prescelta. Se il nuovo indirizzo dello schermo in R4 ha raggiunto questo valore, allora è necessario muovere alla prossima riga prima di continuare con la visualizzazione della stringa.

La linea 49 muove l'attuale indirizzo dello schermo in R4 a R0. La linea 50 muove un byte dal buffer stringa a R1 e autoincrementa l'indirizzo in R3 di uno.La linea 51 addiziona il valore di offset dello schermo (>60) a R1 prima che la linea 52 lo scriva sullo schermo con la routine VSBW. Le linee 53 e 54 incrementano l'indirizzo dello schermo, e decrementano il contatore dei caratteri dopo ogni scrittura. Se il valore in R5 raggiunge zero, il bit di uguaglianza del registro di STATUS viene fissato a uno. La linea 55 controlla questa condizione e dirige la

logica della routine all'etichetta "L1" finchè R5 non è zero. Se R5 è uguale a zero, allora tutti i caratteri della stringa sono stati visualizzati, e la linea 56 ritorna al programma.

La linea 57 confronta i nuovi indirizzi dello schermo da essere usati con il valore in R7. Se R4 è maggiore o uguale a R7, il prossimo byte da essere visualizzato dovrebbe andare alla prossima riga partendo dalla colonna tre. Se R4 è minore di R7, la linea 58 dirige la logica all'etichetta "PLOOP" per scrivere un'altro byte. Le linee 59 e 60 aggiustano l'indirizzo dello schermo, ed i valori limite della riga, per la prossima riga, terza colonna. La linea 61 confronta il valore limite della riga con 766. La riga massima utilizzabile che dovrebbe essere stata selezionata è 24. L'indirizzo della prima colonna alla linea 24 è uguale a 736. Addizionando il valore limite della riga a questo si ha il massimo valore per R7 di 766. Se R7 è maggiore o uguale a 766, lo schermo attuale visualizzato deve fare uno "scroll" in su, prima che il resto della stringa possa essere visualizzato. Finchè questo valore masimo non g stato raggiunto, la linea 62 salterà all'etichetta "PLOOP" per scrivere un'altro byte. La linea 63 esegue un BL a SCROLL quando il valore massimo in R7 è stato raggiunto.

L'effetto dello scroll della visualizzazione sullo schermo è di muovere tutti i valori della tavola schermo in su di una riga e riempire la riga inferiore con degli spazi. Qualunque cosa visualizzata sulla prima riga sarà perduta appena lo scroll dello schermo sarà effettuato.

Per aiutarvi in questo, un buffer di memoria temporaneo viene messo da parte alla linea 5 per mantenere una riga di dati (32 byte). Le linee 67, 68, e 69 caricano i valori iniziali necessari per l'operazione di scroll. I valori dell'indirizzo in VDP RAM in R0 è inizialmente fissato a -32 perchè la linea 70 addiziona 64 ad esso ogni volta attraverso il ciclo chiamato L4. La prima volta attraverso questo ciclo, R0 diventa uguale a 32, l'indirizzo della riga 2. R1 deve contenere l'indirizzo del buffer di linea e R2 la lunghezza, cioè 32 byte.

La linea 71 legge una riga di caratteri dello schermo nel buffer di riga (LBUF). La linea 72 aggiusta l'indirizzo VDP RAM in R0 per 32 e la linea 73 controlla il nuovo valore per vedere se lo scroll è stato fatto. Se il nuovo valore in R0 è uguale a >2E0, la logica salta all'etichetta S1. Altrimenti, la logica continua a NP alla linea 83. La linea 83 scrive il contenuto del buffer di linea al nuovo indirizzo schermo, e la linea 84 salta all'etichetta L4 per completare il ciclo. Questo ciclo continua finchè il confronto alla linea 73 è vero, nel qual caso la logica salta all'etichetta S1. Le linee da 77 a 82 copiano l'indirizzo del buffer di linea, e la sua lunghezza nei registri R13 e R14, e procedono a riempire il buffer di linea con spazi (>20). La linea 82 li mantiene ritornando al'etichetta L3 finchè tutti i 32 byte del buffer di linea siano stati riempiti con spazi. La logica della routine, poi va alla linea 83 la quale scrive i contenuti del buffer di linea (tutti spazi) sull'ultima riga dello schermo.

La logica si trasferisce all'etichetta L4. Il confronto alla linea 73 ora non ha effetto sulla logica di SCROLL, e la linea 76 ritorna alla linea 64 di PRINT. Dopo lo scroll, il limite e i valori dell'indirizzo schermo sono aggiustati per -32 alle linee 64 e 65 prima di continuare a visualizzare il resto della stringa.

Qui inclusi vi sono due programmi che dimostrano l'uso l'uso delle routine ASSEMBLY con il TI BASIC o l'EX-BASIC

DEFINIZIONE DEI CARATTERI ASSEMBLER

Questo programma ridefinisce il set di caratteri standard così che le lettere minuscole appaiano come "VERE" minuscole, cioè, con le parti discendenti. Questo significa che le parti terminali delle lettere p, q, g, j, e y, possono essere stampate sullo schermo. Inoltre, sono stati definiti alcuni caratteri di lingua straniera. Le istruzioni DATA in questa routine usano lo stesso tipo di codice dei caratteri dell'istruzione CALL CHAR del TI BASIC. I caratteri con il codice ASCII da 30 a 126 sono provvisti di definizione. Lo spazio per ridefinire i caratteri da 127 a 143 è stato incluso, così che voi potete usare questa routine per definire velocemente i caratteri grafici all'inizio di un programma BASIC o EX-BASIC.

Notate che questo programma non può essere assemblato se il codice sorgente sia l'EX-BASIC DEF/EQU e l'E/A DEF/REF. Per usare il programma sia con il TI BASIC o l'EX-BASIC, salvare le versioni separate del codice sorgente e assemblare i due file oggetto su due dischi separati. La versione in EX-BASIC dovrebbe essere assemblata con l'opzione "C" (compressa). Il nome del file oggetto dovrebbe essere "DSK1.CHARDF".

Per chiamare la routine dal vostro programma in BASIC o EX-BASIC, voi dovrete usare la seguente routine all'inizio del programma.

```
100 CALL INIT
110 CALL CLEAR
120 CALL LOAD("DSK1.CHARDF")
130 CALL LINK("CHARDF")
```

Se voi usate l'EX-BASIC, voi potete salvare la routine di cui sopra con il nome del file "DSK1.LOAD". Quando voi selezionate l'EX-BASIC, esso partirà automaticamente. Poi qualsiasi programma in EX-BASIC che farete girare avrà i caratteri definiti dalla routine se voi userete un'istruzione CALL LINK("CHARDF"). Una volta che la routine è caricata nell'espansione di memoria, non è necessario ricaricarla, a meno che l'espansione non venga spenta.

```
* DEFINIZIONE CARATTERI ASSEMBLER *
*     di David Migicovsky          *
*     Copyrigt (c) 1983            *
* by Steve davis Publishing        *

LE DUE RIGHE CHE SEGUONO SONO
PER LA VERSIONE IN EX-BASIC

          DEF   CHARDF,VMBW

VMBW      EQU   >2024

LE DUE RIGHE SEGUENTI SONO PER
L'USO CON IL TI BASIC E L'E/A

          DEF   CHARDF
          REF   VMBW
```

```
               IDENTIFICATORE PATTERN          *CARATT. ASCII

NEWDEF DATA >7C7C,>6C6C,>6C6C,>7C7C           *       30
       DATA >0000,>0000,>0000,>0000           *       31
       DATA >0000,>0000,>0000,>0000           * " "   32
       DATA >1010,>1010,>1000,>1000           * "!"   33
       DATA >2828,>2800,>0000,>0000           * """   34
       DATA >2828,>7C28,>7C28,>2800           * "#"   35
       DATA >3854,>5038,>1454,>3800           * "$"   36
       DATA >6064,>0810,>204C,>0C00           * "%"   37
       DATA >2050,>5020,>5448,>3400           * "&"   38
       DATA >0808,>1000,>0000,>0000           * "'"   39
       DATA >0810,>2020,>2010,>0800           * "("   40
       DATA >2010,>0808,>0810,>2000           * ")"   41
       DATA >0028,>107C,>1028,>0000           * "*"   42
       DATA >0010,>107C,>1010,>0000           * "+"   43
       DATA >0000,>0000,>0030,>1020           * ","   44
       DATA >0000,>007C,>0000,>0000           * "-"   45
       DATA >0000,>0000,>0030,>3000           * "."   46
       DATA >0004,>0810,>2040,>0000           * "/"   47
       DATA >3844,>4444,>4444,>3800           * "0"   48
       DATA >1030,>1010,>1010,>3800           * "1"   49
       DATA >3844,>0408,>1020,>7C00           * "2"   50
       DATA >3844,>0418,>0444,>3800           * "3"   51
       DATA >0818,>2848,>7C08,>0800           * "4"   52
       DATA >7C40,>7804,>0444,>3800           * "5"   53
       DATA >1820,>4078,>4444,>3800           * "6"   54
       DATA >7C04,>0810,>2020,>2000           * "7"   55
       DATA >3844,>4438,>4444,>3800           * "8"   56
       DATA >3844,>443C,>0408,>3000           * "9"   57
       DATA >0030,>3000,>3030,>0000           * ":"   58
       DATA >0000,>3030,>0030,>1020           * ";"   59
       DATA >0810,>2040,>2010,>0800           * "<"   60
       DATA >0000,>7C00,>7C00,>0000           * "="   61
       DATA >2010,>0804,>0810,>2000           * ">"   62
       DATA >3844,>0408,>1000,>1000           * "?"   63
       DATA >3844,>5C54,>5C40,>3800           * "@"   64
       DATA >3844,>447C,>4444,>4400           * "A"   65
       DATA >7824,>2438,>2424,>7800           * "B"   66
       DATA >3844,>4040,>4044,>3800           * "C"   67
       DATA >7824,>2424,>2424,>7800           * "D"   68
       DATA >7C40,>4078,>4040,>7C00           * "E"   69
       DATA >7C40,>4078,>4040,>4000           * "F"   70
       DATA >3C40,>405C,>4444,>3800           * "G"   71
       DATA >4444,>447C,>4444,>4400           * "H"   72
       DATA >3810,>1010,>1010,>3800           * "I"   73
       DATA >0404,>0404,>0444,>3800           * "J"   74
       DATA >4448,>5060,>5048,>4400           * "K"   75
       DATA >4040,>4040,>4040,>7C00           * "L"   76
       DATA >446C,>5454,>4444,>4400           * "M"   77
       DATA >4464,>6454,>4C4C,>4400           * "N"   78
       DATA >7C44,>4444,>4444,>7C00           * "O"   79
       DATA >7844,>4478,>4040,>4000           * "P"   80
       DATA >3844,>4444,>5448,>3400           * "Q"   81
       DATA >7844,>4478,>5048,>4400           * "R"   82
       DATA >3844,>4038,>0444,>3800           * "S"   83
       DATA >7C10,>1010,>1010,>1000           * "T"   84
       DATA >4444,>4444,>4444,>3800           * "U"   85
       DATA >4444,>4428,>2810,>1000           * "V"   86
       DATA >4444,>4454,>5454,>2800           * "W"   87
       DATA >4444,>2810,>2844,>4400           * "X"   88
       DATA >4444,>2810,>1010,>1000           * "Y"   89
       DATA >7C04,>0810,>2040,>7C00           * "Z"   90
       DATA >0810,>3844,>7C40,>3800           * "["   91
       DATA >3030,>3FFF,>FE7C,>180C           * "\"   92
```

```
        DATA >2010,>3844,>7C40,>3800           * "]" 93
        DATA >3844,>4040,>4438,>1000           * "^" 94
        DATA >1028,>0038,>4848,>3400           * "_" 95
        DATA >0000,>3840,>4038,>1000           * "`" 96
        DATA >0000,>3848,>4848,>3400           * "a" 97
        DATA >6020,>3824,>2424,>7800           * "b" 98
        DATA >0000,>3844,>4044,>3800           * "c" 99
        DATA >0C08,>3848,>4848,>3C00           * "d" 100
        DATA >0000,>3844,>7C40,>3800           * "e" 101
        DATA >1824,>2070,>2020,>2000           * "f" 102
        DATA >0000,>3C44,>3C04,>0438           * "g" 103
        DATA >6020,>2834,>2424,>2400           * "h" 104
        DATA >1000,>7010,>1010,>7C00           * "i" 105
        DATA >0800,>1808,>0848,>4830           * "j" 106
        DATA >2020,>2428,>3028,>2400           * "k" 107
        DATA >3010,>1010,>1010,>7C00           * "l" 108
        DATA >0000,>A854,>5454,>5400           * "m" 109
        DATA >0000,>5824,>2424,>2400           * "n" 110
        DATA >0000,>3844,>4444,>3800           * "o" 111
        DATA >0000,>7824,>2438,>2020           * "p" 112
        DATA >0000,>3048,>3808,>080C           * "q" 113
        DATA >0000,>5824,>2020,>2000           * "r" 114
        DATA >0000.>3C40.>3804.>7800           * "s" 115
        DATA >2020,>7820,>2024,>1800           * "t" 116
        DATA >0000,>4848,>4848,>3400           * "u" 117
        DATA >0000,>4444,>2828,>1000           * "v" 118
        DATA >0000,>D454,>5454,>2800           * "w" 119
        DATA >0000,>4428,>1028,>4400           * "x" 120
        DATA >0000,>4444,>3C04,>0418           * "y" 121
        DATA >0000,>7C48,>1024,>7C00           * "z" 122
        DATA >1820,>2040,>2020,>1800           * "{" 123
        DATA >1010,>1000,>1010,>1000           * "|" 124
        DATA >3008,>0804,>0808,>3000           * "}" 125
        DATA >0000,>2054,>0800,>0000           * "~" 126
        DATA >0000,>0000,>0000,>0000           *     127
        DATA >0000,>0000,>0000,>0000           *     128
        DATA >0000,>0000,>0000,>0000           *     129
        DATA >0000,>0000,>0000,>0000           *     130
        DATA >0000,>0000,>0000,>0000           *     131
        DATA >0000,>0000,>0000,>0000           *     132
        DATA >0000,>0000,>0000,>0000           *     133
        DATA >0000,>0000,>0000,>0000           *     134
        DATA >0000,>0000,>0000,>0000           *     135
        DATA >0000,>0000,>0000,>0000           *     136
        DATA >0000,>0000,>0000,>0000           *     137
        DATA >0000,>0000,>0000,>0000           *     138
        DATA >0000,>0000,>0000,>0000           *     139
        DATA >0000,>0000,>0000,>0000           *     140
        DATA >0000,>0000,>0000,>0000           *     141
        DATA >0000,>0000,>0000,>0000           *     142
        DATA >0000,>0000,>0000,>0000           *     143

CHARDF  LI R0,1008 *THIS IS A DECIMAL NUMBER
        LI R1,NEWDEF
        LI R2,904  *THIS IS A DECIMAL NUMBER
        BLWP @VMBW
        RT
        END
```

DEFINIZIONE CARATTERI CON LA M/M

Per usarlo con l'assembler linea per linea e la M/M, cancellate tutte le linee eccetto le istruzioni DATA. Cambiate l'etichetta nella prima linea DATA (NEWDEF) con "ND". Inserite due nuove linee alloinizio del programma, mettendo la prima istruzione DATA alla linea tre.

```
          AORG  >7D00
VM        EQU   >6028
ND        DATA  >7C7C,....
```

Immediatamente dopo l'ultima linea di DATA, addizionate le seguenti linee.

```
CD        LI    R0,1008
          LI    R1,ND
          LI    R2,904
          BLWP  @VM
          CLR   @>837C
          B     *R11
          END
```

Per accedere alla routine da un programma BASIC, voi dovreste usare l'istruzione CALL LINK("CD")

PROGRAMMA "BAR GRAPH"

Questo programma esempio è stato creato da due appassionati del TI-99/4A dell'AUSTRALIA, i sigg. PHILL WEST e BERNIE ELSNER.

E una routine progettata per essere chiamata da qualsiasi programma in TI BASIC con i moduli M/M o l'E/A inseriti. La routine permette di tracciare in alta risoluzione Barre di Grafici in vari colori. Per accedere alla routine usare l'istruzione BASIC CALL LINK ("BGRAPH",COLUMN,COLOR,HEIGHT).

COLUMN dovrebbe essere un numero tra 1 e 28, la colonna dello schermo nel quale la barra apparirà. COLOR è il colore della barra, un numero da 1 a 7, come qui sotto indicato.

NUMERO DEL COLORE	COLORE	SET DI CARATTERI USATI
1	NERO	10
2	BLU SCURO	11
3	ROSSO SCURO	12
4	GIALLO SCURO	13
5	VERDE SCURO	14
6	MAGENTA	15
7	BIANCO	16

HEIGHT dovrebbe essere un numero tra 1 e 160, indicando con questo, quante righe/pixel sarà alta la barra. Tutte le barre vengono disegnate iniziando dalla riga 20, e possono estendersi verso l'alto, fino alla prima riga. Questo lascia libere le righe dalla 21 fino alla 24 per il testo da visualizare.

Di seguito al listato ASSEMBLY vi è un programma dimostrativo in TI BASIC che vi permette di vederne i risultati. Se voi vorrete usare il modulo E/A, dovrete caricare il file di supporto BASIC dal dischetto "A" dell'E/A. Il programma dimostrativo vi suggerisce di di mettere il dischetto nel drive 1. Il codice oggetto del programma ASSEMBLY BAR GRAPH dovrebbe essere salvato sul dischetto con il nome del file di "BGRAP/O" (per Bar Graph Oggetto), e dovrebbe essere sul drive 1 quando fate girare il "Demo".

```
* ROUTINE BAR-GRAPH
* PER L'USO CON LA M/M DAL BASIC
* DI PHIL WEST E BERNIE ELSNER

           DEF   BGRAPH
           REF   VMBW,VSBW,NUMREF,XMLLNK,ERR

D1         DATA  >0000      * Definizione dei caratteri
           DATA  >0000      *
           DATA  >0000      *
           DATA  >003C      *
           DATA  >3C3C      *
           DATA  >3C3C      *
           DATA  >3C3C      *
           DATA  >3C3C      *
D2         DATA  >1040      * Byte dei colori
           DATA  >60AQ      *
```

```
          DATA  >C0D0      *
          DATA  >F000      *
BGRAPH    CLR   R0
*
* Ottiene i tre parametri dalla lista LINK
*
          LI    R1,>0001   * Ottiene il primo parametro.
          BLWP  @NUMREF
          BLWP  @XMLLNK
          DATA  >1200
          MOV   @834A,R3
          CI    R3,>0000   * Controlla se il valore è valido.
          JGT   C
          B     @E
C         CI    R3,>001D
          JLT   F
          B     @E
F         INC   R1         * Ottiene il secondo parametro.
          BLWP  @NUMREF
          BLWP  @XMLLNK
          DATA  >1200
          MOV   @>834A,R4
          CI    R4,0000    * Controlla se il valore è valido.
          JGT   G
          B     @E
G         CI    R4,>0008
          JLT   H
          B     @E
H         INC   R1         * Ottiene il terzo parametro.
          BLWP  @NUMREF
          BLWP  @XMLLNK
          DATA  >1200
          MOV   @>834A,R5
          CI    R5,0000    * Controlla se il valore è valido.
          JGT   J
          B     *R11
J         CI    R5,>00A1
          JLT   K
          B     @E
*
```



```
* Definisce i caratteri
*
K         LI    R2,>0008   * Quanti byte da scrivere, e inc.per R0.
          LI    R0,>0640   * Scrive l'indirizzo in VDP
B         LI    R1,D1      * Indirizzo in R1 dei DATA dei caratteri.
A         BLWP  @VMBW     * Scrive 8 byte nella tavola dei caratteri
          A     R2,R0      * Incrementa di 8 per il pross. carattere
          INC   R1         * Si sposta in sotto di una riga DATA.
          CI    R1,D1+8    * Ultimo indirizzo DATA?
          JLT   A          * No, ricomincia da "A"
          CI    R0,>0800   * Ultimo carattere da definire?
          JLT   B          * No, fare il prossimo carattere
```

```
* Definisce il set colori dei caratteri
*
        LI     R0,>0319   * Indirizzo VDP per il set caratteri N°10
        LI     R1,D2      * Set dei colori
        LI     R2,>0007   * 7 byte da scrivere
        BLWP   @VMBW
*
* Determina il carattere da usare
*
        SLA    R4,3       * Moltiplica per 8
        AI     R4,>00C0   * 192 meno l'offset di 96 = 96+8 = 104,
*                           che è il primo carattere del set N°10
        SWPB   R4         * Carattere da usare
        LI     R6,>0200   * Decrementa il valore della riga
        LI     R0,>0261   * Fissa la riga 20 e colonna 2
        A      R3,R0      * Colonna da usare
*
* Disegna la barra, un pixel alla volta.
*
N       CLR    R7         * Azzerra il contatore
        MOV    R4,R1      * Si prepara a scrivere il 1° byte
M       BLWP   @VSBW     * Scrive un byte
        DEC    R5         * Riduce la barra di una riga di pixel
P       CI     R5,>0000   * Finito?
        JGT    L          * No, continua
        B      *R11       * Si, ritorna al BASIC
L       AI     R1,>0100   * Ottiene il prossimo carattere
        INC    R7         * Incrementa il contatore
        CI     R7,>0000   * Scrito un carattere intero?
        JLT    M          * No, continua
        S      R6,R0      * Si, decrementa la riga
        JMP    N          * Continua con la prossima riga
*
* Parametro d'errore
*
E       LI     R0,>1300   * Errore, valore sbagliato
        BLWP   @ERR
        END
```

Ed ecco il programma chiamante in BASIC.

```
100 REM  DEMO BAR GRAPH
110 REM  TI BASIC CON M/M O
120 REM  MODULO E/A
130 REM  BY PHIL WEST E BERNIE ELSNER
140 CALL CLEAR
150 RANDOMIZE
160 PRINT "METTI IL FILE 'BGRAPH/O' NEL"
170 PRINT "DISK DRIVE 1, POI":"PREMI ENTER"
180 INPUT E$
190 PRINT "CARICO IL LINGUAGGIO MACCHINA ":"PREGO ATTENDERE"
200 CALL INIT
210 CALL LOAD("28706,0,0,0,0,0,0,0,0")
220 CALL LOAD("DSK1.BGRAPH/O")
230 PRINT "CHE MODULO USI?":"INSERIRE"
240 PRINT " *E* PER L'E/A, *M* PER LA M/M"
250 INPUT E$
260 IF E$=M THEN 350
270 IF E$<>E THEN 250
280 PRINT "METTI IL DISCO "A" DELL'E/A"
290 PRINT "NEL DRIVE 1, POI":"PREMI ENTER"
300 INPUT E$
310 CALL CLEAR
320 CALL LOAD("DSK1.BSCSUP")
330 REM RISERVA SPAZIO PER LA DEFINIZIONE DEI CARATTERI
340 REM NELLA ROUTINE IN LINGUAGGIO MACCHINA
350 FOR I=104 TO 159
360 CALL CHAR(I,"")
370 NEXT I
380 CALL CLEAR
390 CALL SCREEN(15)
400 PRINT " PROFITTI 1982-84"
410 PRINT " (Q)UIT O (R)IPETERE"
420 F=1
430 FOR I=1 TO 28
440 F=F*1,195
450 CALL LINK("BGRAPH",I,1,F)
460 NEXT I
470 FOR D=1 TO 800
480 NEXT D
490 FOR I=1 TO 28
500 CALL VCHAR(1,I+2,32,20)
510 CALL LINK("BGRAPH",I,6,F)
520 F=F/1,195
530 NEXT I
540 FOR D=1 TO 800
550 NEXT D
560 CALL HCHAR(1,1,32,640)
570 REM GENERA PARAMETRI CASUALI
580 FOR I=1 TO 28
```

```
590 B=INT(RND*7+1)
600 C=INT(RND0+1)
610 REM DISEGNA LE BARRE CON LA ROUTINE IN L/M
620 CALL LINK("BGRAPH",I,B,C)
630 NEXT I
640 CALL KEY(0,K,S)
650 IF K=82 THEN 380
660 IF K<>81 THEN 640
670 END
```

MANUALE DI RIFERIMENTO E/A

I seguenti riferimenti vi forniranno altre informazioni su come unire l'ASSEMBLY con il BASIC.

Leggere queste sezioni.

Dalla sezione 17-1 pag. 273 alla sezione 17-2-6 pag. 289
Sezione 18-2-5 pag. 300
Sezione 21-1 pag. 326
Dalla sezione 24-4 pag. 410 alla sezione 24-4-9 pag. 418
Dalla sezione 24-11 pag. 440 alla sezione 24-11-3 pag. 442

GUARDARE QUESTI TERMINI SUL GLOSSARIO

GPL
Loader (caricatore)
Utilità
Workspace (spazio di lavoro, o spazio registri)
Workpace pointer register (puntatore ai registri workspace)

QUALCHE CONSIGLIO...PRIMA DI LASCIARCI

Ci sono parecchi punti, a parte l'effettiva comprensione del linguaggio ASSEMBLY del TMS9900, che possono aiutarvi a programmare in questo linguaggio un po più facilmente, ed in maniera più produttiva. Questo implica certe procedure, e abitudini di lavoro, ed apprezzano il lavoro fatto a mano quando si sta creando un programma ASSEMBLY.

Non importa fino a che punto voi siete, o potete diventare esperti con la programmazione ASSEMBLY, non è quasi mai pratico o sensato iniziare battendo semplicemente le linee di codice di un nuovo programma. Sebbene questo possa essere possibile con il BASIC, ciò non è consigliabile con l'ASSEMBLY. Una volta che voi avete deciso l'applicazione che desiderate scrivere nel linguaggio del TMS9900, cogliete il momento giusto per delineare il vostro programma con carta e matita, prima di mettervi al computer. Come sottolineato all'inzio di questo libro, il linguaggio ASSEMBLY richiede una grande quantità di dettagli per ogni piccola cosa che voi avrete bisogno di fare.

Consultate qualsiasi materiale di riferimento che avete a vostra disposizione per assicurarvi che il primo abbozzo del vostro programma sia il più possibile libero da errori, sia di sintassi che di logica. Fate uso delle parti di programma che avete scritto in precedenza. Molti programmatori professionisti tengono un'inventario delle subroutine che essi hanno sviluppato, e che possono essere riassemblate modularmente per formare la base di applicazioni in nuovi programmi.

Ad un livello molto fondamentale, voi dovreste fare molta attenzione alla struttura e progetto dei vostri programmi. I linguaggi ASSEMBLY sono come base, libere forme di linguaggio. Finch'è le regole della sintassi vengono rispettate, le direttive, le istruzioni e dati possono apparire in qualsiasi ordine. Questo è uno di quei casi dove la libertà non è certamente una buona cosa. I programmi esempio dati in questo libro seguono tutti un modello o un formato prescritto. Le direttive, i dati delle istruzioni, riferimenti, ed uguaglianze sono tutte elencate prima. Queste sono poi seguite dalle effettive istruzioni del programma. Tanto quanto è possibile i vostri programmi dovrebbero incorporare uno schema "Top Down" = (dall'alto al basso). Come voi leggete i programmi dall'alto al basso, cosi dovrete seguire la logica sequenza degli avvenimenti come essi si succedono.

Se voi state per creare un quadro, una sinfonia, o un grattacielo, progettare e pianificare con molta cura è importante per completare con pieno successo la vostra creazione. Lo stesso criterio viene applicato creando un programma ASSEMBLY. Voi avrete bisogno di considerare in anticipo come affrontare l'incarico. Voi dovete anticipare le necessità del programma stesso, come un temporaneo campo di memorizzazione per i calcoli e manipolazione dei dati. Controllare le funzioni ripetitive che possono essere codificate come subroutine. Documentate il vostro programma con commenti ed etichette che spiegano la sua logica. Allineate coerentemente le etichette, istruzioni ed operandi sul limite della colonna che essi occuperanno. Questo renderà il vostro programma più facile da leggere. Fate molta attenzione alla chiarezza, perchè normalmente rende gli errori più facili da scoprire.

Sviluppate un vostro proprio stile nel progettare i programmi, ed attenetevi ad esso. Prendete in prestito liberamenteda altri esempi che trovate da sostituire per,

o apportare delle migliorie al vostro progetto. Voi potete imparare moltissimo dagli altri programmatori ASSEMBLY. Un buon sistema per trovare questo tipo di aiuto è quello di associarsi con altri utenti della stessa zona, possibilmente più esperti. E, ricordatevi, che uno dei vostri migliori insegnanti è semplicemente: PROVARE e SBAGLIARE.

Molti prodotti sono disponibili per aiutarvi nello sviluppo e "Debugging" (ricerca e correzione degli errori). Come menzionato, questi includono calcolatori HEX speciali, e package di software come la serie della TEXAS "AIUTO ALLA PROGRAMMAZIONE I-II-III. Esistono anche disassemblatori che riconvertono il codice oggetto nel codice sorgente. Sia l'assembler linea per linea, che l'E/A hanno un programma interattivo di "debugging" che vi assiste nella vostra programmazione. Professionisti ed hobbysti traggono vantaggi da questi prodotti di aiuto. Comunque, bisogna sottolineare che nessuno di questi è un sostituto per una fondamentale comprensione dei principi del linguaggio ASSEMBLY.

Ricordate di fare sempre delle copie di Back Up dei programmi e dei file di dati, prima di far girare i programmi, o accedere ai dati. Questo può salvarvi dal mal di testa e dalla frustrazione. Un computer che lavora bene abitualmente, diventa più critico quando lavora nel linguaggio ASSEMBLY.

Se voi state usando l'E/A, una stampante è una delle periferiche più utili che vi possiate avere. I listati che voi farete con lei sono un'aiuto prezioso per il "debugging" dei programmi, e creano una documentazione che più di una volta vi verrà in aiuto per controllare il programma. Questo richiede di leggere il programma una linea alla volta. Come voi incontrate ogni linea di codice, prendete nota di cosa e stabilito, e registrate il risultato come voi lo capite su un pezzo di carta. Per esempio, disegnate un gruppo di colonne e segnate la parte alta di ciascuna con il nome dell'etichetta o registro implicato. Poi, come i valori vengono inizializzati o cambiati, registrate questi valori nella colonna corretta. Tracciate una linea attraverso il valore precedente, quando questo cambia. Non cancellatelo, perchè poi potreste avere bisogno di riferirvi a lui in un secondo tempo. Prendete nota del numero di linea che ha causato il cambio. Questo vi mostrerà i contenuti di ogni registro o campo ad ogni passo. Se voi incontrate una istruzione per sommare i contenuti di due registri con valori che sono stati determinati 20 linee prima, i valori da essere sommati saranno nelle corrispondenti colonne da voi create.

Sviluppate buone abitudini di lavoro, con disciplina, e imparate a fare attenzione ai dettagli che vi aiuteranno nella programmazione in linguaggio ASSEMBLY, e in tutti gli altri tipi di interazione con il computer. "LA PRATICA RENDE PERFETTI". Più voi Lavorate con l'ASSEMBLY e più bravi diventerete con esso. Naturalmente, questo richiede qualche sforzo e sopratutto determinazione da parte vostra. Sarete però ricompensati con un'accresciuta comprensione del vostro computer, ed un potente strumento che mette il suo potenziale, cioè il linguaggio ASSEMBLY del TMS9900, al vostro comando.

PROGRAMMA ESEMPIO DELLA PAG. 21

```
   1-------10---------20---------30---------40--------50

01          DEF    START
02          REF    VSBW,VMBW
03 STATUS   EQU    >837C
04 SAVRTN   DATA   >0000
05 AMTX     DATA   >000A
06 AMTY     DATA   >0021
07 DECTEN   DATA   >000A
08 HEX30    DATA   >0030
09 PNTANS   BSS    >2
10 WSPREG   BSS    >20
11 START    LWPI   >WSPREG
12          MOV    R11,@SAVRTN
13          BL     @CLEAR
14 ADDUP    A      @AMTX,@AMTY
15          MOV    @AMTY,R5
16          CLR    R4
17          DIV    @DECTEN,R4
18 MASKUP   A      @HEX30,R5
19          MOV    R5,@PNTANS
20          MOV    R4,R5
21          CLR    R4
22          DIV    @DECTEN,R4
23          A      @HEX30,R4
24          SLA    R5,8
25          MOVB   R5,@PNTANS
26 PUTUP    LI     R0,738
27          LI     R1,PNTANS
28          LI     R2,2
29          BLWP   @VMBW
30 EOJ      MOV    @SAVRTN,R11
31          CLR    R0
32          MOVB   R0,@STATUS
33          RT
34 CLEAR    CLR    R0
35          CLR    R1
36 LOOP     BLWP   @VSBW
37          CI     R0,767
38          JEQ    CLEARX
39          INC    R0
40          JMP    LOOP
41 CLEARX   B      *R11
42          END
```

```
 0001                     DEF START
 0002                     REF VSBW,VMBW
 0003      837C  STATUS   EQU   >837C
 0004 0000 0000  SAVRTN   DATA  >0000
 0005 0002 000A  AMTX     DATA  >000A
 0006 0004 0021  AMTY     DATA  >0021
 0007 0006 000A  DECTEN   DATA  >000A
 0008 0008 0030  HEX30    DATA  >0030
 0009 000A       PNTANS   BSS   >2
 0010 000C       WSPREG   BSS   >20
 0011 002C C80B  START    MOV   R11,@SAVRTN
      002E 0000'
 0012 0030 02E0           LWPI  WSPREG
      0032 000C'
 0013 0034 06A0           BL    @CLEAR
      0036 007E'
 0014 0038 A820  ADDUP    A     @AMTX,@AMTY
      003A 0002'
      003C 0004'
 0015 003E C160           MOV   @AMTY,R5
      0040 0004'
 0016 0042 04C4           CLR   R4
 0017 0044 3D20           DIV   @DECTEN,R4
      0046 0006'
 0018 0048 A160  MASKUP   A     @HEX30,R5
      004A 0008'
 0019 004C C805           MOV   R5,@PNTANS
      004E 000A'
 0020 0050 C144           MOV   R4,R5
 0021 0052 04C4           CLR   R4
 0022 0054 3D20           DIV   @DECTEN,R4
      0056 0006'
 0023 0058 A120           A     @HEX30,R4
      005A 0008'
 0024 005C 0A85           SLA   R5,8
 0025 005E D805           MOVB  R5,@PNTANS
      0060 000A'
 0026 0062 0200  PUTUP    LI    R0,738
      0064 02E2
 0027 0066 0201           LI    R1,PNTANS
      0068 000A'
 0028 006A 0202           LI    R2,2
      006C 0002
 0029 006E 0420           BLWP  @VMBW
      0070 0000
 0030 0072 C2E0  EOJ      MOV   @SAVRTN,R11
      0074 0000'
 0031 0076 04C0           CLR   R0
 0032 0078 D800           MOVB  R0,@STATUS
      007A 837C
 0033 007C 045B           RT
 0034 007E 04C0  CLEAR    CLR   R0
 0035 0080 04C1           CLR   R1
 0036 0082 0420  LOOP     BLWP  @VSBW
      0084 0000
 0037 0086 0280           CI    R0,767
      0088 02FF
 0038 008A 1302           JEQ   CLEARX
 0039 008C 0580           INC   R0
```

```
 99/4 ASSEMBLER
VERSION 1.2                                                            PAGE 0002
  0040 008E 10F9          JMP   LOOP
  0041 0090 045B  CLEARX  B     *R11
  0042                    END
```

C'é quí la Tavola dei Simboli costruita dal programma. Ogni simbolo usato accanto al suo indirizzo, é mostrato in ordine alfabetico. L'indirizzo dopo ciascun Simbolo puó essere quello effettivo, (STATUS a >837C), o il valore del contatore di locazione.

```
 99/4 ASSEMBLER
VERSION 1.2                                                            PAGE 0003
  ' ADDUP    0038    ' AMTX     0002    ' AMTY     0004    ' CLEAR    007E
  ' CLEARX   0090      DECTEN   0006    ' EOJ      0072    ' HEX30    0008
  ' LOOP     0082    ' MASKUP   0048    ' PNTANS   000A    ' PUTUP    0062
    R0       0000      R1       0001      R10      000A      R11      000B
    R12      000C      R13      000D      R14      000E      R15      000F
    R2       0002      R3       0003      R4       0004      R5       0005
    R6       0006      R7       0007      R8       0008      R9       0009
  ' SAVRTN   0000    D START    002C      STATUS   837C    E VMBW     0070
  E VSBW     0084    ' WSPREG   000C
  0000 ERRORS
```

PROGRAMMA DI "SORT" A PAG. 79

```
001              DEF    SORT
002              REF    VMBW,VSBW,VWTR,VMBR,KSCAN,DSRLNK,VSBR
003 KEYVAL       EQU    >8375
004 STATUS       EQU    >837C
005 PABIN        EQU    >F80
006 PABOUT       EQU    >FA0
007 RECBUF       EQU    >1000
008 PNTR         EQU    >8356
009 CLOSB        BYTE   >01
010 WRITB        BYTE   >03
011 READB        BYTE   >02
012 ENTV         BYTE   >0D
013 LEFTV        BYTE   >08
014 RITEV        BYTE   >09
015 CURVAL       BYTE   >1E
016 REDOV        BYTE   >06
017 QUITV        BYTE   >05
018              EVEN
019 CURPAT       DATA   >007E,>4242,4242,7E00
020 BORDER       DATA   >FFFF,>2020,>2020,>2020
021              DATA   >2020,>2020,>2020,>2020
022              DATA   >2020,>2020,>2020,>2020
023              DATA   >0202,>2020,>2020,>FFFF
024 ERRMSG       TEXT   ' I/O ERROR ! CODE=   '
025 TIT1         TEXT   ' Chapter 10 Sort Program    '
026 TIT2         TEXT   ' Output File/Device Name?   '
027 TIT3         TEXT   '       Sorting Now          '
028 TIT4         TEXT   '        End Of Job          '
029 TIT5         TEXT   '  Press Any Key To Continue '
030 TOT1         TEXT   '       Record Count         '
031 TOT2         TEXT   '======Input===Output========'
032 ORD1         TEXT   ' <A>scend    <D>escend   >  '
033 EQLN         TEXT   '============================'
034 WATFLD       TEXT   ' What Field # To Sort On >  '
035 FIELD1       TEXT   ' First Name--------------1  '
036 FIELD2       TEXT   ' Last  Name--------------2  '
037 VERIFY       TEXT   ' Screen Complete - Redo?    '
038 EOF          DATA   >0000
039 SAVRTN       DATA   >0000
040 FLDLEN       DATA   >000E
041 TBFEND       DATA   >000D
042 TABEND       DATA   >000F
043 TABLEN       DATA   >0010
044 BLINK        DATA   >0BA0
045 DTEN         DATA   >000A
046 RELREC       DATA   >0000
047 TBMARK       DATA   >FFFF
048 INFILE       DATA   >0005,RECBUF,>5050,>0000,>000A
049              TEXT   'DSK1.FILE1                  '
050 OUTFLE       DATA   >0003,RECBUF,>5050,>0000,>0000
051              TEXT   'DSK                         '
052 OPTION       BSS    2
053 INBUFF       BSS    80
054 FNAME        EQU    INBUFF
055 LNAME        EQU    INBUFF+14
056 OUTBUF       BSS    80
057 HLDTAB       BSS    >10
058 MYREG        BSS    >20          059 SRTTAB BSS >1000
```

```
060 SORT        MOV    R11,@SAVRTN  Salva l'indirizzo di ritorno
061             LWPI   MYREG        Carica il puntatore al workspace
062             LI     R0,>0766     mette il colore del bordo al Rosso scuro
063             BLWP   @VWTR
064             LI     R0,>0380     Mette il set dei caratteri come Bianco/Bl
065             LI     R1,>F400
066 LOOP1       BLWP   @VSBW
067             INC    R0
068             CI     R0,>039F     Mette il carattere del bordo al Rosso
069             JLT    LOOP1
070             LI     R1,>6600
071             BLWP   @VSBW
072             LI     R0,>08F0     Carica il pattern del cursore
073             LI     R1,CURPAT
074             LI     R2,8
075             BLWP   @VMBW
076 PROMPT      BL     @SCREEN
077             LI     R0,66
078             LI     R1,TIT1
079             BLWP   @VMBW        Visualizza il titolo del programma
080             LI     R0,130
081             LI     R1,TIT2      Prompt per il dispositivo di output
082             BLWP   @VMBW
083             LI     R0,195
084             LI     R10,15
085             BL     @CURSOR      Ottiene il nome del dispositivo di output
086             MOV    R7,@@OUTFLE+8 Muove il descrit. lungh. del file al PAB
087             LI     R0,195
088             LI     R1,OUTFLE+10
089             MOV    R7,R2
090             BLWP   @VMBR        Legge il nome dispositivo nel PAB
091 WHTFLD      LI     R0,258       Quale campo da classificare?
092             LI     R1,WATFLD
093             LI     R2,28
094             BLWP   @VMBW
095             LI     R0,290
096             LI     R1,FIELD1
097             BLWP   @VMBW
098             LI     R0,322
099             LI     R1,FIELD2
100             BLWP   @VMBW
101             LI     R10,1
102             LI     R0,282
103             BL     @CURSOR      Ottiene la scelta del N° campo
104             CI     R9,>3100     Verifica se la risposta è = a "1" o "2"
105             JLT    WHTFLD
106             CI     R9,>3200
107             JGT    WHTFLD       Altrimenti, ancora il prompt
108 ORDER       LI     R0,418       Prompt per l'ordine "Ascend/Discend"?
109             LI     R1,ORD1
110             BLWP   @VMBW
111             LI     R10,1
112             LI     R0,443
113             BL     @CURSOR      Ottiene la scelta dell'ordine
114             CI     R9,>4100     Verifica se la risposta è = "A" o "D"
115             JLT    ORDER
116             JEQ    VERIF
117             CI     R9,>4400
118             JNE    ORDER        Altrimenti, di nuovo il prompt
119 VERIF       LI     R0,706
120             LI     R1,VERIFY
121             BLWP   @VMBW        Verifica le scelte fatte
122             CLR    R10
123             BL     @CURSOR
124             LI     R0,443
125             BLWP   @VSBR
```

```
126          MOVB  R1, @OPTION    Salva l'ordine dell'opzione
127          LI    R0,282
128          BLWP   @VSBR
129          MOVB  R1, @OPTION+1 Salva il campo opzione
130          LI    R0,PABIN     Stabilisce il PAB file input
131          LI    R1,INFILE
132          LI    R2,20
133          BLWP   @VMBW
134          BL     @DSRIN       Apre in modo input
135          MOVB   @READB,R1
136          BLWP   @VSBW        Mette la lettura del'OP/CODE  nel PAB+0
137          BL     @SCREEN      Ritorna allo schermo grafico
138          LI    R0,226
139          LI    R1,TOT1      Visualizza l'intestazione del cont. record
140          BLWP   @VMBW
141          LI    R1,258
142          LI    R0,TOT2
143          BLWP   @VMBW
144          LI    R8,SRTTAB    Destina l'indirizzo per muovere la Word a Rt
145          MOV    @FLDLEN,R9   Lunghezza del campo per muovere la Word a R
146          CLR    @EOF
147 READ     LI    R0,PABIN+6   Indirizzo PAB del N° record
148          LI    R1,RELREC
149          LI    R2,2
150          BLWP   @VMBR        Ottiene il numero del record relativo
151          MOV    @RELREC,R4
152          LI    R3,295
153          LI    R0,300
154          BL     @FIGUR       Visualizza il contatore di record input
155          BL     @DSRIN       Legge un record
156          MOV    @EOF,@@EOF    Controlla per la fine del file
157          JNE   DONE1        Se è EOF vai a "DONE1"
158          LI    R0,RECBUF    Indirizzo del buffer VDP RAM
159          LI    R1,INBUF     Indirizzo del buffer CPU RAM
160          LI    R2,80        Lunghezza del record
161          BLWP   @VMBR        Ottiene un record
162          LI    R1,>3100     Controlla per quale campo
163          CB    R1,@@OPTION+1
164          JNE   MOVLNM
165          LI    R7,FNAME     Indirizzo del primo nome per MOVWRD
166          JMP   MOVS
167 MOVLNM   LI    R7,LNAME     Indirizzo dell'ultimo nome per MOVWRD
168 MOVS     BL     @MOVWRD      Eseguire MOVWRD - Carica la tavola del sort
169          MOV    @RELREC,*R8+ Carica il N° del record relat. nella tavola
170          JMP   READ         Legge un altro record
171 DONE1    MOV    @TBMARK,*R8  Carica la fine della tavola indicatrice
172          LI    R0,322
173          LI    R1,EQLN
174          LI    R2,28
175          BLWP   @VMBW        Visualizza "Adesso classifico"
176          LI    R0,354
177          LI    R1,TIT3
178          BLWP   @VMBW
179          LI    R5,>4400     Carica R4 con il valore di "D"
180 COMPAR   LI    R1,SRTTAB    Indirizzo della prima entrata nella tavola
181 GETTAB   MOV   R1,R3        Muove il valore dell'indirizzo da R1 a R3
182          MOV   R1,R2        Muove il valore dell'indirizzo da R1 a R2
183          A      @TBFEND,R2   Calcola la lunghezza dell'entrata nella tavol
184          MOV   R3,R4
185          A      @TABLEN,R4   Calcola l'indir. della pross. entr. tavola
186          C      @TBMARK,*R4  Controlla per la fine della tavola
187          JEQ   DONE2        Se è la fine della tavola, vai a DONE2
188          MOV   R3,R0        Salva l'indirizzo di "A" in R0
189          MOV   R4,R1        Salva l'indirizzo di "B" in R1
190 ORDCHK   CB    R5, @OPTION   Controlla per Ascend/Descend
191          JEQ   CLOPD
```

```
*
*  ORDINE ASCENDENTE *
*
192 CLOP        C      R3,R2         Controlla per l'ultimo byte del campo
193             JEQ    GETTAB        Se =, ottiene la prossima entrata della tavo
194             CB     *R3+,*R4+     Confronta un byte di "A" con "B"
195             JEQ    CLOP          Se è uguale, ripetere
196             JGT    SWIT          Se A > B, interrompere
197             JMP    GETTAB        Se A < B, allora ottenere la prossima entrat
*
*  ORDINE DISCENDENTE *
*
198 CLOPD       C      R3,R2         Controlla per l'ultimo byte del campo
199             JEQ    GETTAB        Se uguale ottiene la prossima entrata
200             CB     *R3+,*R4+     Confronta un byte di "A" con "B"
201             JEQ    CLOPD         Se è uguale, ripetere
202             JGT    GETTAB        Se A > B allora ottiene la prossima entrata
203 SWIT        MOV    R1,R7         Carica R7 con l'indirizzo di "B"
204             LI     R8,HLDTAB     Carica R8 con l'indirizzo HLDTAB
205             MOV    @TABLEN,R9     Carica R9 con la lunghezza da muovere
206             BL      @MOVWRD       Muove "B" nell'area di appoggio
207             MOV    R0,R7         Carica l'indirizzo di "A"d
208             MOV    R1,R8         Carica l'indirizzo di "B"
209             BL      @MOVWRD       Muove "A" a "B"
210             LI     R7,HLDTAB     Carica l'indirizzo di appoggio
211             MOV    R0,R8         Carica l'indirizzo di "A"
212             BL      @MOVWRD       Muove l'appoggio di (B) ad "A"
213             JMP    COMPAR        Ritorna indietro e control. la seq. della ta
214 MOVWRD      MOV    R9,R6         Carica la lunghezza da muovere
215             A      R7,R6         Calcola il massimo indirizzo
216 MOVEM       MOV    *R7+,*R8+     Muove una Word, e incrementa l'indirizzo
217             C      R7,R6         Controlla per l'indirizzo massimo
218             JNE    MOVEM         Se non è il massimo, muovi ancora
219             RT                   Altrimenti, ritorna
220 DONE2       CLR    R9            Pulisce il registro 9
221             LI     R0,PABOUT     Stabilisce il file output nel PAB
222             LI     R1,OUTFLE
223             LI     R2,25
224             BLWP   @VMBW
225             BL     @DSROUT        Apre il file output
226             MOVB   @WRITE,R1      Mette l'op-code della scrittura nel PAB+0
227             BLWP   @VSBW
228             LI     R0,PABIN      Ristabilisce il file input nel PAB
229             LI     R1,INFILE
230             BLWP   @VMBW
231             BL     @DSRIN         Apre il file input
232             MOVB    @READB,R1     Mette l'op-code della lettura nel PAB+0
233             BLWP   @VSBW
234             LI     R8,SRTTAB+14   Indirizzo nella tav. del N° del rec.rel.
235 GETRR       MOV    R8,R4         Muove l'indirizzo in R8 a R4
236             S      @FLDLEN,R4     Calcola l'indirizzo della prima entrata
237             C      @TBMARK,*R4    Controlla per la fine della tavola
238             JEQ    EOJ           Se è la fine della tavola, vai a EOJ
239             MOV    R8,R1         Muove l'indirizzo del N° del rec. rel. a R1
240             LI     R0,PABIN+6    Carica R0 con l'indirizzo di destinazione
241             LI     R2,2          Carica R2 con la lunghezza dei dati
242             BLWP    @VMBW         Mette il N° del record rel. nel PAB+6
243             BL      @DSRIN        Legge questo record
244             INC    R9            Incrementa il contatore dei record output
245             BL      @DSROUT       Scrive un record output
246             A       @TABLEN,R8    Calcola l'indirizzo del prossim rec. relati
247             MOV    R9,R4         Muove il contatore dei record a R4
248             LI     R3,304
249             LI     R0,309
250             BL      @FIGUR        Visualizza il contatore dei record output
251             JMP    GETRR         Ottiene il prossimo N° del record relativo
```

```
252 EOJ         LI      R0,354          Visualizza "END OF JOB" = ( fine del lavoro)
253             LI      R1,TIT4
254             LI      R2,28
255             BLWP     @VMBW
256             LI      R0,706
257             LI      R1,TIT5
258             BLWP     @VMBW
259             LI      R0,PABIN
260             MOVB     @CLOSB,R1       Mette l'op-code di close nel PAB+0
261             BLWP     @VSBW
262             LI      R0,PABOUT        Mette l'op-code di close nel PAB+0
263             BLWP     @VSBW
264             BL       @DSRIN          Chiude il file input
265             BL       @DSROUT         Chiude il file output
266 EOJX        CLR     R10
267             BL       @CURSOR         Esegue un "PRESS ANY KEY"
268 EOJQ        CLR      @STATUS         Pulisce il byte dello Status GPL
269             MOV      @SAVRTN,R11    Muove l'indirizzo di ritorno in R11
270 TX          RT                      Ritorna (B *R11)
271 CURSOR      CLR     R9              Pulisce il registro 9
272             CLR      @KEYVAL         Pulisce l'indir. del valore del tasto prem.
273             CLR      @STATUS         Pulisce il byte dello status GPL
274             MOV     R10,R10         Controlla R10 per il valore di zero
275             JEQ     CURL1           Se è zero, vai al LOOP1 cursore
276             MOV     R0,R8           Salva l'indirizzo di inizio del cursore
277             A       R8,R10          Calcola l'indirizzo massimo del cursore
278             MOV     R8,R7           Muove l'indirizzo iniziale all'accumulatore
279 CURPUT      CLR     R6              Pulisce il registro 6
280             MOV     R7,R0           Muove l'accumulatore in R0
281             LI      R1,>2000        Carica il registro 1 con lo spazio
282             BLWP     @VSBW          Visualizza lo spazio
283 CURL1       BLWP     @KSCAN         Esegue la scansione della tastiera
284             MOVB     @STATUS, @STATUS Controlla il tasto battuto
285             JNE     DECTET          Se il tasto è premuto, quale tasto?
286             AI      R6,4            Addiziona 4 al registro
287             C       R6, @BLINK       Confronta R6 con il contatore BLINK
288             JLT     CURL1           Se minore, ripeti CURL1
289             MOV     R10,R10         Controla R10 con il valore zero
290             JEQ     CURL1           Se è zero, vai a CURL1
291             CLR     R6              Pulisce il registro 6
292             MOVB     @CURVAL,R1     Muove il codice del cursore in R1
293             BLWP     @VSBW          Visualizza il cursore
294 CURL2       INC     R6              Addiziona 1 a R6
295             C       R6, @BLINK      Confronta R6 con il contatore di BLINK
296             JLT     CURL2           Se è minore, ripeti CURL2
297             JMP     CURPUT          Ripetere il ciclo del cursore
298 DETECT      CB       @REDOV, @KEYVAL Controlla per il valore di "REDO"
299             JEQ     REDOX           Se REDO, vai all'uscita REDO
300             CB       @QUITV, @KEYVAL Controlla per il valore di "QUIT"
301             JEQ     EOJQ            Se QUIT, vai a EOJ/QUIT
302             MOV     R10,R10         Controlla R10 con il valore zero
303             JEQ     XT              Se è zero, ritorna
304             CB      @ENTV, @KEYVAL   Controlla per il valore di "ENTER"
305             JEQ     ENTER
306             CB      @LEFTV, @KEYVAL   Controlla per la freccia sinistra
307             JEQ     LEFT
308             CB       @RITEV, @KEYVAL  Controlla per la freccia destra
309             JEQ     RITE
310             C       R7,R10          Controlla per il massimo indirizzo del curs.
311             JEQ     CURPUT          Se è uguale, vai a "CURPUT"
312             MOV     R7,R0           Muove il nuovo indirizzo in R0
313             MOVB    @KEYVAL,R1       Muove il valore del tasto premuto in R1
314             MOVB    @KEYVAL,R9       Salva il valore del tasto premuto in R9
315             BLWP    @VSBW            Visualizza il carattere del tasto premuto
316             INC     R7              Addiziona 1 all'accumulatore dell'indirizzo
317             JMP     CURPUT          Vai a "CURPUT"
```

```
318 LEFT      C     R7,R8        Controlla per il minimo indirizzo del cursore
319           JEQ   CURPUT       Se è uguale, vai a "CURPUT"
320           MOV   R7,R0        Muove all'indirizzo corrente in R0
321           LI    R1,>2000     Carica R1 con il codice del caratt. spazio
322           BLWP  @VSBW         Scrive uno spazio all'indirizzo corrente
323           DEC   R7           Sottrae 1 dall'indirizzo corrente
324           JMP   CURPUT       Vai a "CURPUT"
325 RITE      C     R7,R10       Controlla per l'indirizzo massimo del cursore
326           JEQ   CURPUT       Se è uguale,vai a "CURPUT"
327           INC   R7           Addiziona 1 all'indirizzo corrente
328           JMP   CURPUT       Vai a "CURPUT"
329 ENTER     LI    R1,>2000     Carica R1 con il codice del carattere spazio
330           MOV   R7,R0        Muove l'indirizzo corrente in R0
331           BLWP  @VSBW         Lo spazio sopra il simbolo del cursore
332           S     R8,R7        Calcola l'efettiva lunghezza dei dati
333           JEQ   CURSOR       Se è zero, ripeti il ritorno al cursore
334           RT                 Altrimenti ritorna
335 REDOX     B     @PROMPT      Si dirama all'indirizzo di "PROMPT"
336 SCREEN    CLR   R0           Pulisce il registro 0
337           LI    R1,BORDER    Carica R1 con l'indirizzo del bordo
338           LI    R2,32        Carica R2 con la lunghezza dei dati
339 SCRL      BLWP  @VMBW         Scrive una linea del pattern
340           CI    R0,736       Confronta R0 con il massimo valore
341           JHE   LEAVE        Se = o >, vai a "LEAVE"

342           AI    R0,32        Addiziona 32 a R0
343           JMP   SCRL         Vai a "SCRL"
344 LEAVE     LI    R0,2         Carica R0 con il valore di 2
345           LI    R1,EQLN      Carica R1 con l'indirizzo grafico
346           LI    R2,28        Carica R2 con la lunghezza della linea
347           BLWP  @VMBW        Visualizza all'indirizzo dello schermo 2
348           LI    R0,738       Carica R0 con il valore 738
349           BLWP  @VMBW        Visualizza all'indirizzo 738 dello schermo
350 RTN       RT                 Ritorna
351 DSRIN     LI    R6,PABIN+9   Carica R6 con l'indirizzo PAB
352           JMP   DSR
353 DSROUT    LI    R6,PABOUT+9  Carica R6 con l'indirizzo PAB
354 DSR       MOV   R6,@PNTR     Muove R6 all'indirizzo del puntatore
355           BLWP  @DSRLNK      esegue un ritorno al dispositivo di servizio
356           DATA  8
357           JNE   RTN          Se nessun errore, ritorna
358 ERROR     MOV   R6,R0        Muove R6 in R0
359           AI    R0,-8        Calcola l'indirizzo del byte 1 del PAB
360           BLWP  @VSBR        Legge il byte 1 del PAB in R1
361           SRL   R1,5         Sposta R1 a destra di 5 posizioni
362           MOV   R1,@EOF      Muove R1 nel "FLAG"dell'EOF
363           CB    @QUITV,R1    Controlla per EOF (valore di >05)
364           JEQ   RTN          Se è uguale, ritorna
365           BL    @SCREEN      Altrimenti, rifare lo schermo grafico
366           MOV   R6,R0        Muove R6 in R0
367           INC   R0           Calcola l'indirizzo del byte 10 del PAB
368           LI    R1,INBUFF    Carica R1 tempor. nell'indirizzo in CPU RAM
369           LI    R2,20        Carica R2 con la lunghezza da scrivere
370           BLWP  @VMBR        Ottiene il nome del file in ERROR
371           LI    R0,259       Carica R0 con l'indirizzo di schermo 259
372           BLWP  @VMBW        Visualizza il nome del file in ERROR
373           MOV   @EOF,R1      Muove il codice di errore in R1
374           AI    R1,>3000     Crea un numero ASCII
375           LI    R0,343       Indirizzo di schermo per il codice di errore
376           BLWP  @VSBW        Visualizza il codice di errore
377           LI    R0,322       Indirizzo di schermo per il messaggio d'errore
378           LI    R1,ERRMSG    indirizzo del messaggio
379           BLWP  @VMBW        Visualizza il messaggio di errore
380           B     @EOJX        Vai a EOJ
381 FIGUR     MOV   R4,R5        Muove R4 in R5
382           CLR   R4           Pulisce R4
```

```
383            DIV    @DTEN,R4      Divide R4/R5 di dieci
384            AI     R5,>30        Crea un numero ASCII in R5
385            SLA    R5,8          Sposta R5 a sinistra di 8 posti
386            MOV    R5,R1         Muove R5 in R1
387            BLWP   @VSBW         Visualizza una cifra
388            DEC    R0            Sottrae 1 da R0
389            C      R0,R3         Confronta R0 con R3
390            JNE    FIGUR         Se non è uguale, ripeti
391            RT                   Altrimenti, ritorna
392            END
```

```
0001            DEF   GO
0002            REF   VSBW,VMBW,VMBR,NUMREF,XMLLNK,STRREF,ERR
0003  FPAC      EQU   >834A
0004  SBUFF     BSS   256                 Definisce il buffer di stringa
0005  LBUF      BSS   32                  Definisce il buffer di linea
0006  LIM       DATA  >0001,>001B,>001C Definisce l'area delle righe e colonne
0007  GO        MOV   R11,R10             Salva l'indirizzo di ritorno
0008            CLR   R0                  Pulisce R0 - NUMREF Param. N°1
0009            LI    R1,1                Inizializza R1 con 1 - NUMREF Param. N°2
0010            BL    @GETNUM              Ottiene la prima variabile (numero riga
0011            BL    @CHKLMR              Controlla i limiti della riga
0012            MOV   @FPAC,R4             Muove il valore della riga in R4
0013            DEC   R4                  Aggiusta per l'ASSEMBLY
0014            SLA   R4,5                Moltiplica la riga per 32
0015            MOV   R4,R7               Muove l'indirizzo riga in R7
0016            INC   R1                  Incrementa R1 per il prossimo param.
0017            BL    @GETNUM              Ottiene la seconda variabile (numero Col)
0018            BL    @CHKLMC              Controlla i limiti della colonna
0019            A     @FPAC,R4             Addiziona il valore riga
0020            INC   R4                  Aggiusta per il BASIC
0021            INC   R1                  Incrementa R1 per il prossimo parametro
0022            LI    R2,SBUF             Carica R2 con l'indirizzo del buffer stringa
0023            SETO  @SBUF                Inizializza la prima Word con >FFFF
0024            BLWP  @STRREF              Ottiene la terza variabile (la stringa)
0025            CLR   R5                  Pulisce R5 (il contatore dei byte)
0026            MOV   R2,R3               Muove l'indirizzo di SBUF in R3
0027            MOVB  *R3+,R5             Ottiene il primo byte (lunghezza stringa)
0028            JEQ   XT                  Se la lunghezza stringa è uguale a zero, SALTA A XT
0029            SWPB  R5                  Cambia i byte di R5 (aggiusta a sinistra
0030            BL    @PRINT               Visualizza la stringa
0031  XT        B     *R10                Ritorna al programma chiamante
0032  GETNUM    BLWP  @NUMREF              Ottiene il numero base (in virgola mobile)
0033            BLWP  @XMLLNK              Esegue la conversione Virg. Mob./intero
0034            DATA  >1200               Indirizzo della routine XML
0035            B     *R11                Ritorna
0036  CHKLMC    C     @FPAC, @LIM+4          Confronta l'intero con 28
0037            JGT   ERROR               Se è maggiore, salta a "ERROR"
0038            JMP   CHK                 Salta a "CHK"
0039  CHKLMR    C     @FPAC, @LIM+2          Confronta l'intero con 24
0040            JGT   ERROR               Se è maggiore, salta a "ERROR"
0041  CHK       C     @FPAC, @LIM            Confronta l'intero con 1
0042            JLT   ERROR               Se è minore, salta a "ERROR"
0043            B     *R11                Ritorna
0044  ERROR     LI    0,>1300             Carica R0 col valore del messaggio d'errore
0045            BLWP  @ERR                 Si dirama alla routine del messag. d'errore
0046  PRINT     MOV   R11,9               Salva l'indirizzo di allacciamento
0047            LI    R6,>6000            Carica R6 con il valore offset dello schermo
0048            AI    R7,30               Calcola il prossimo indirizzo di riga
0049  PLOOP     MOV   R4,R0               Muove l'indirizzo del VDP in R0
0050            MOVB  *R3+,R1             Ottiene un byte dal buffer di stringa
0051            AB    R6,R1               Addiziona l'offset dello schermo a R1
0052            BLWP  @VSBW                Scrive un byte
0053            INC   R4                  Punta all'indirizzo del prossimo schermo
0054            DEC   R5                  Decrementa il contatore dei caratteri
```

```
0055            JNE   L1           Se R5 non è uguale a 0, salta a L1
0056            B     *R9          Ritorna
0057  L1        C     R4,R7        Confronta il nuovo indirizzo con il Limite
0058            JL    PLOOP        Se è minore, salta a "PLOOP"
0059            AI    R7,32        Altrimenti incremen. il limite di 1 riga
0060            AI    R4,4         Incrementa l'indirizzo dello schermo di 4
0061            CI    R7,766       è la nuova riga dello schermo
0062            JLE   PLOOP        Salta a PLOOP
0063            BL    @SCROLL      Si dirama a SCROLL
0064            AI    R7,-32       Aggiusta il limite dopo lo scroll
0065            AI    R4,-32       Aggiusta l'indirizzo schermo dopo lo scroll
0066            JMP   PLOOP        Salta a PLOOP
0067  SCROLL    LI    R0,-32       Inizializza l'indirizzo dello schermo
0068            LI    R1,LBUF      Carica R1 con l'indirizzo buffer di lineA
0069            LI    R2,32        Carica R2 con la lunghezza della linea
0070  L4        AI    R0,64        Si muove sotto di una linea
0071            BLWP  @VMBW        Legge una linea nel buffer di linea
0072            AI    R0,-32       Si muove in su di una linea
0073            CI    R0,>2E0      è questa l'ultima linea?
0074            JLT   NP           Se non la è, salta a "NP"
0075            JEQ   S1           Se la è, salta a "S1"
0076            B     *R11         Lo scroll è fatto, ritorna
0077  S1        MOV   R1,R13       Copia il puntatore del buffer
0078            MOV   R2,R14       Copia la lunghezza del buffer
0079            LI    R15,>2020    Carica due spazi in R15
0080  L3        MOV   R15,*R13+    Muove gli spazi nel buffer
0081            DECT  R14          Decrementa il contatore dei byte
0082            JNE   L3           Prossima Word in PAD
0083  NP        BLWP  @VMBW        Scrive più byte
0084            JMP   L4           Salta a R4
0085            END
0086
```